

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ АНАЛІЗУ
МЕРЕЖНОГО ТРАФІКУ НА ПРИСУТНІСТЬ АТАК ТИПУ DDOS
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR
NETWORK TRAFFIC ANALYSIS TO DETECT DDOS ATTACKS

Виконав: студент 2 курсу, групи КНТ-129м
Спеціальності

121 Інженерія програмного забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Сергієнко О.С.

(прізвище та ініціали)

Керівник Федорончак Т. В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Сергієнко Олександр Сергійович
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження та програмна реалізація методів аналізу мережного трафіку на присутність атак типу DDoS. Research and Software Implementation of Methods for Network Traffic Analysis to Detect DDoS Attacks

керівник проекту (роботи) Федорончак Тетяна Василівна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “03” листопада 2020 року № 301

2. Строк подання студентом проекту (роботи) 14 грудня 2020 року

3. Вихідні дані до проекту (роботи) рекомендована література, завдання на дипломний проект

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми. 6. Економіко-організаційна частина. 7. Охорона праці та безпека в надзвичайних ситуаціях.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Федорончак Т.В., доцент		
6 Економіко-організаційна частина	Пожуєва Т.О., професор		
7 Охорона праці та безпека в надзвичайних ситуаціях	Коробко О.В., ст. викладач		
Нормоконтроль	Андреев М.О., асистент		

7. Дата видачі завдання “ 4 ” вересня 2020 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4–5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Розробка економіко-організаційної частини.	10 тиждень	Розділ 6
8	Розробка заходів з охорони праці та безпеки в надзвичайних ситуаціях.	11 тиждень	Розділ 7
9	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
10	Нормоконтроль та рецензування.	14–15 тижні	
11	Захист роботи.	16 тиждень	

Студент _____ Сергієнко О.С.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Федорончак Т.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 156 с., 11 табл., 77 рис., 2 дод., 31 джерел.

КОМП'ЮТЕРНІ МЕРЕЖІ, МЕРЕЖНИЙ ТРАФІК, АНАЛІЗ МЕРЕЖНОГО ТРАФІКУ, DDOS, LINUX, C++, LIBPCAP.

Об'єкт дослідження – методи виявлення атак типу DDoS.

Предмет дослідження – комп'ютерні програми для аналізу мережного трафіку на присутність атак типу DDoS.

Мета роботи – створення програмної системи для аналізу мережного трафіку на присутність атак типу DDoS.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування C++, бібліотека libpcap, інструмент CMake, середовище Microsoft Visual Studio Code, інструмент генерування трафіку abccir, засіб автоматизації процесів у Unix-подібних операційних системах bash, система контролю версій Git.

Результати. Створено системне програмне забезпечення, що здійснює аналіз мережного трафіку на присутність атак типу DDoS.

Висновки. Досліджено та програмно реалізовано алгоритми виявлення аномального мережного трафіку. Розроблено програмну систему для виявлення атак типу DDoS.

Галузь використання – захист та моніторинг мережі на присутність атак типу DDoS.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 156 pages, 11 tables, 77 figures, 2 appendixes, 31 sources.

COMPUTER NETWORKS, NETWORK TRAFFIC, NETWORK TRAFFIC ANALYSIS, DDOS, LINUX, C++, LIBPCAP.

Object of study are methods of detecting DDoS attacks.

Subject of study are computer programs for network traffic analysis on the presence of the DDoS attacks.

The purpose of the work is to create a software system for network traffic analysis on the presence of the DDoS attacks.

Materials, methods, and tools: structured and object-oriented programming, C++ programming language, libpcap, CMake, IDE MS Visual Studio Code, traffic generation tool abcip, bash-scripts, Git.

Results. A software for network traffic analysis on the presence of the DDoS attacks is created.

Conclusions. Algorithms of detection network traffic for the anomaly presence were researched and implemented. A software system for detection of the DDoS attacks was developed.

The field of use is a network security and monitoring of the DDoS attacks.

ЗМІСТ

	С.
ВСТУП.....	9
1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ.....	11
1.1 Опис предметної області.....	11
1.2 Основні види DoS-атак та їхні ефекти.....	14
1.2.1 Атаки, спрямовані на мережний стек.....	14
1.2.2 Атаки, спрямовані на сервер.....	16
1.2.3 Атаки, спрямовані на сервіс.....	20
1.3 Методи протидії DDoS атакам	22
1.4 Огляд засобів для здійснення DDoS атак	23
1.5 Огляд алгоритму виявлення атак типу DDoS на основі розрахунку ентропії Шеннона.....	25
1.6 Існуючі програмні засоби що вирішують аналогічні завдання.....	29
2 МАТЕРІАЛИ І МЕТОДИ.....	30
2.1 Дослідження алгоритму виявлення аномалій мережного трафіку на основі розрахунку кумулятивної суми.....	30
2.2 Дослідження алгоритму виявлення аномалій мережного трафіку на основі розрахунку інформаційної ентропії на часових інтервалах.....	32
2.3 Дослідження та розробка алгоритму виявлення UDP флуду.....	36
2.4 Дослідження та розробка схеми роботи системи як цілісного програмного забезпечення.....	42
3 ОПИС ПРОГРАМИ.....	44
3.1 Структура та функціонал розроблюваної програми	44
3.2 Аналіз програмних засобів.....	50
3.2.1 Обґрунтування використання мови програмування C++.....	50
3.2.2 Обґрунтування використання бібліотеки libpcap.....	51
3.2.3 Обґрунтування використання генератора сценаріїв складання CMake.....	51

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ.....	53
4.1 Детальний опис компонентів системи.....	53
4.2 Опис потокової архітектури системи.....	60
5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ.....	62
5.1 Керівництво системного програміста.....	62
5.1.1 Призначення програми та системні вимоги.....	62
5.1.2 Характеристики програми.....	62
5.1.3 Підготовка програми до запуску.....	65
5.1.4 Вхідні і вихідні дані.....	65
5.2 Керівництво оператора.....	65
5.2.1 Призначення програми.....	65
5.2.2 Умови виконання програми.....	66
5.2.3 Встановлення програми.....	66
5.2.4 Видалення програми.....	66
5.2.5 Виконання програми.....	67
5.3 Експериментальні дослідження.....	69
5.4 Експериментально отримані результати дослідження.....	74
5.5 Аналіз, узагальнення та обґрунтування достовірності отриманих результатів.....	77
6 ЕКОНОМІКО-ОРГАНІЗАЦІЙНА ЧАСТИНА.....	80
6.1 Концепція економічного обґрунтування.....	80
6.2 Визначення складу команди.....	82
6.3 Планування етапів розробки продукту.....	82
6.4 Визначення витрат на розробку.....	84
6.4.1 Розрахунок основної заробітної плати.....	84
6.4.2 Розрахунок додаткової заробітної плати.....	84
6.4.3 Розрахунок відрахувань на соціальні заходи.....	85
6.4.4 Розрахунок матеріальних витрат та витрат по споживанню послуг.....	85

6.4.5 Розрахунок витрат по амортизації.....	86
6.4.6 Визначення інших операційних витрат.....	87
6.4.7 Визначення сукупності операційних витрат.....	88
6.4.8 Визначення повної собівартості продукту.....	89
6.5 Визначення ціни реалізації.....	89
6.6 Визначення очікуваного доходу.....	90
6.7 Визначення фінансового результату.....	91
6.8 Висновки щодо економічної ефективності продукту.....	91
7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ.....	94
7.1 Аналіз потенційних небезпек.....	94
7.2 Заходи із забезпечення безпеки.....	95
7.3 Заходи з забезпечення виробничої санітарії та гігієни праці.....	97
7.4 Заходи з забезпечення безпеки у надзвичайних ситуаціях.....	102
7.4.1 Заходи з пожежної безпеки.....	102
7.4.2 Заходи з цивільного захисту.....	105
ВИСНОВКИ.....	109
ПЕРЕЛІК ПОСИЛАНЬ.....	111
ДОДАТОК А Текст програми.....	114
ДОДАТОК Б Слайди презентації.....	145

ВСТУП

На сьогоднішній день, мережа Інтернет нараховує понад чотири з половиною мільярда активних користувачів, і ця кількість постійно зростає щороку. Так само й зростає залежність бізнесу і окремих осіб від постійного та швидкого доступу до віддалених сервісів (банків, магазинів, інформаційних джерел, тощо). Збільшується кількість розподіленої інформації, доступ до якої можна отримати швидко та з будь-якої точки світу. Проте така тенденція має один негативний аспект – це безпека та надійність. Конфіденційні данні викрадаються, а віддалені ресурси – блокуються в результаті здійснення мережесих атак, що призводить до, подекуди, величезних збитків та в решті решт незручності для кінцевих користувачів.

Мережесі атаки, спрямовані на блокування роботи віддаленого ресурсу (веб-сервісу або сайту), відносять до групи атак впливу на доступність. Використовуючи широкий спектр векторів нападу (різні види флуду), такі атаки є однією з найсерйозніших загроз для ресурсів в Інтернеті. Кажучи про них, говорять про атаки типу DoS (англ. denial-of-service – відмова в обслуговуванні). Якщо така атака здійснюється координовано, з використанням певної кількості робочих станцій, то її називають розподіленою – DDoS (distributed denial-of-service).

Інформація щодо масштабних DDoS атак обов'язково висвітлюється в новинах, а сама атака призводить до значних фінансових збитків, втрати довіри користувача та негативно впливає на імідж компанії, сервіси котрої, в результаті, були заблоковані на довготривалий час. Щодня реєструються тисячі фактів нових DoS атак. Не всі вони є подіями світового масштабу, проте навіть невелика за потужністю атака здатна вивести з ладу сайт середньої за розмірами компанії. Взагалі, кожна компанія яка має власний веб-сервіс або сайт, постійний доступ до якого є критичним, знаходиться у групі ризику і має прийняти міри протидії.

Протидія атакам типу DDoS має бути комплексною – починаючи від надійної мережесі архітектури, закінчуючи координованими та оперативними

діями системних адміністраторів. Проте в основі успіху лежить своєчасне виявлення факту присутності атаки. Саме тому програмне забезпечення виявлення присутності атак типу DDoS є необхідною умовою побудови ефективної стратегії зменшення наслідків атаки, розробка якого є метою цієї роботи.

Розроблене програмне забезпечення має працювати у режимі фаєрволу між зовнішньою та внутрішньою мережею, постійно аналізуючи поточний мережний трафік на присутність атаки типу DDoS та, за можливості, блокувати його. У разі виявлення аномальної мережної активності відразу виводити відповідне повідомлення.

Також, розроблене ПЗ має бути стійким до самої атаки й обробляти трафік якомога швидше для зменшення негативного впливу на пропускну спроможність мережі в нормальному режимі роботи. Для забезпечення можливості контролю трафіку в ручному режимі має бути присутня функціональність запису оброблених пакетів у вигляді логів, а для корекції роботи користувачем – конфігурація.

Виходячи із вимог, розроблений аналізатор мережного трафіку на присутність атак типу DDoS має представляти собою системне програмне забезпечення для платформи Linux, яке працює в режимі мережевого мосту між двома мережними інтерфейсами та аналізує увесь трафік між ними.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ

ДОСЛІДЖЕННЯ

1.1 Опис предметної області

У даній роботі досліджується проблема виявлення аномалій мережного трафіку, які вказують на присутність атак типу DDoS.

Атака на відмову в обслуговуванні (англ. denial-of-service, DoS) – це напад на комп'ютерну систему через мережу з наміром зробити комп'ютерні ресурси недоступними легітимним користувачам, для яких вона була призначена. Для здійснення атаки, нападник починає відсилати велику кількість мережних запитів чи просто даних на комп'ютер-мішень, використовуючи обчислювальні ресурси своєї робочої станції для вичерпання ресурсів мішені [1].

DoS-атака є атакою типу «точка-точка», тобто єдиними її учасниками є атакуюча станція та комп'ютер-мішень. Наведена нижче схема ілюструє типову DoS-атаку (рис. 1.1).



Рисунок 1.1 – Схема DoS-атаки

Проте за умови великої пропускної спроможності мережі або потужності мішені, DoS-атака може бути неефективною. Для підвищення ефективності атаки роблять розподіленою.

Розподілена атака на відмову в обслуговуванні (англ. distributed denial-of-service, DDoS) – це скоординована DoS-атака, яка відбувається одночасно з великої кількості робочих станцій (з великої кількості IP-адрес) задля збільшення її

ефективності. Такій атаці значно складніше протидіяти. Робочі станції, які беруть у ній участь, називають ботнетом, якщо вони використовуються не легітимно, тобто, без відому їхніх власників. Як правило, це можливо через розповсюдження шкідливого програмного забезпечення, здатного захопити управління робочою станцією. Якщо ж участь у атаці добровільна та здійснюється користувачем станції свідомо, то говорять про флешмоб або ж сплановану атаку. Також, необхідна кількість робочих станцій для здійснення атаки може бути спеціально підготовлена організатором завчасно, без необхідності залучення третіх осіб. Існують навіть спеціальні сервіси що надають послуги оренди ботнету для здійснення атак [1].

Зазвичай ботнетом керує один чи декілька серверів управління, які отримують команду на початок атаки від ініціатора. Таким чином, справжнього ініціатора атаки знайти важко, адже команди ботнету віддає сервер управління. Загальну схему DDoS-атаки зображено на рис. 1.2.

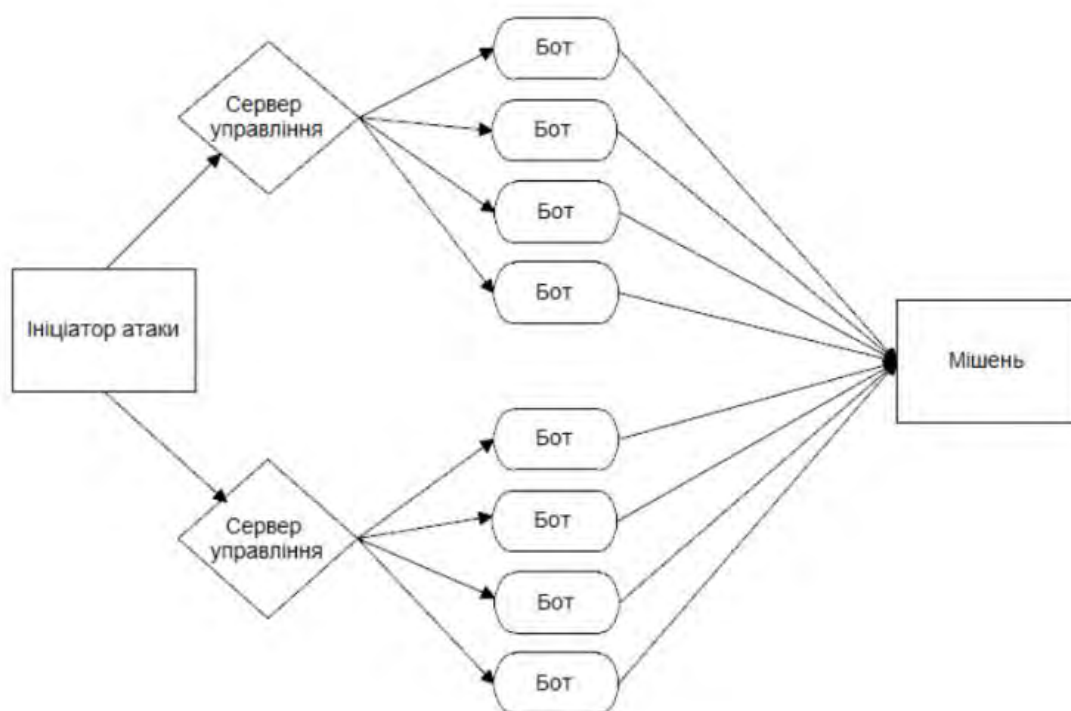


Рисунок 1.2 – Схема DDoS-атаки

Аналіз мережного трафіку – це сукупність дій з перехоплення та обробки мережного трафіку призначеного для інших робочих станцій. Для цього використовують спеціалізоване програмне забезпечення-аналізатор мережного трафіку, який може здійснювати прослуховування мережі. Під'єднання аналізатора до мережі найчастіше здійснюється увімкненням на прослуховування мережного інтерфейсу або увімкненням напряду в розрив каналу [2].

У даній роботі прослуховування та аналіз мережного трафіку є головною задачею розробленого ПЗ. Використовується перший спосіб – прослуховування мережного інтерфейсу.

Мережний міст – це апаратно-програмний або тільки програмний компонент мережі, призначений для об'єднання сегментів комп'ютерної мережі між собою [3]. Міст працює на каналному рівні моделі OSI, тому його базова функціональність зводиться лише до пропуску мережного трафіку між двома сегментами мережі. Загальну схему роботи мережного моста наведено на рис. 1.3.

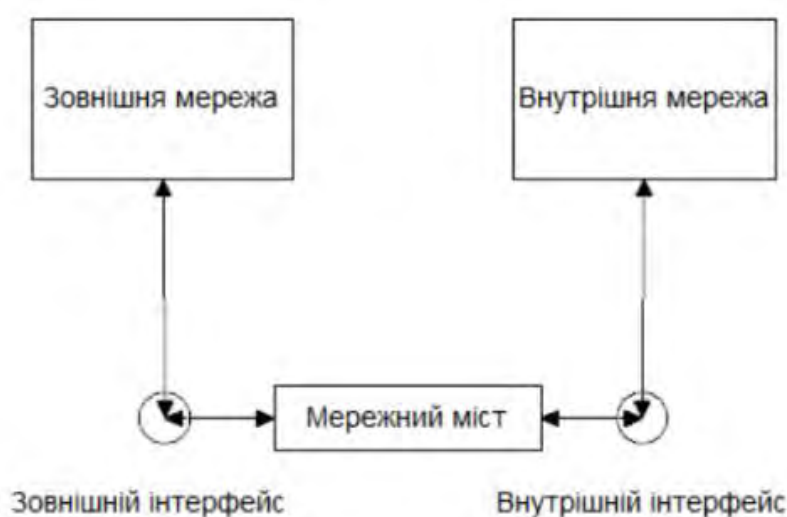


Рисунок 1.3 – Схема роботи мережного моста

Більшість систем активного аналізу трафіку на присутність атак різного типу з можливістю блокування працюють у вигляді мосту між вхідним (зовнішнім) та вихідним (внутрішнім) мережними інтерфейсами (рис. 1.3).

Розроблене ПЗ теж відноситься до систем такого типу, тому використовує пари вхідних-вихідних мережних інтерфейсів для роботи.

1.2 Основні види DoS-атак та їхні ефекти

Атака типу DoS, по своїй сутності, характеризується багатовекторністю та постійністю. Під багатовекторністю розуміється значна кількість можливих напрямків розвитку атаки, оскільки існує декілька способів зайняти ресурси обчислювальної системи або мережі. Під постійністю розуміється значний проміжок часу за який атака відбувається. Взагалі, для того щоб ефект став помітним, може знадобитися певний час, в залежності від потужності атакованої системи та атакуючої сторони, але як тільки атака припиняється, система відновлюється дуже швидко. Тому самоціллю для атакуючого є підтримка DoS умови значний проміжок часу, в період якого ціль не зможе працювати у нормальному режимі [4].

В залежності від обраного вектору ураження, атаки поділяються на три категорії: атаки спрямовані на сервіс як додаток, атаки спрямовані на сервер та атаки спрямовані на мережний стек [4].

1.2.1 Атаки, спрямовані на мережний стек

Атаки цієї категорії мають на меті зменшення пропускнуєї спроможності мережі шляхом відправлення великої кількості трафіку, спрямованого на мережну інфраструктуру об'єкта атаки. Трафік зазвичай є легітимним, але у великій кількості. Атаки такого виду називають флудом (англ. flood). Такі атаки є досить простими в плані генерування трафіку та не потребують значних обчислювальних ресурсів для їх здійснення, проте є не менш ефективними. Зазвичай нападник використовує ботнет з тисяч робочих станцій, які починають відправляти на IP-адресу атакованого серверу величезну кількість пакетів одночасно, в результаті

чого, запити легітимних користувачів не можуть бути оброблені через велике навантаження на мережу [4].

UDP флуд (англ. UDP flood) – це різновид флуду як DoS атаки, в основі якого лежить генерування великої кількості UDP-трафіку, спрямованого на атакований сервер. User Datagram Protocol (UDP) – це протокол транспортного рівня моделі OSI, який не потребує встановлення сесії між клієнтом та сервером (англ. connectionless), що дає змогу відправляти велику кількість пакетів дуже швидко [5]. Така атака не використовує ніякі вразливості роботи мережних протоколів і є легітимним трафіком, проте у великій кількості, що в решті решт призводить до перенасичення мережі та відмові в обслуговуванні [4]. Загальна схема UDP флуду наведена на рис. 1.4.

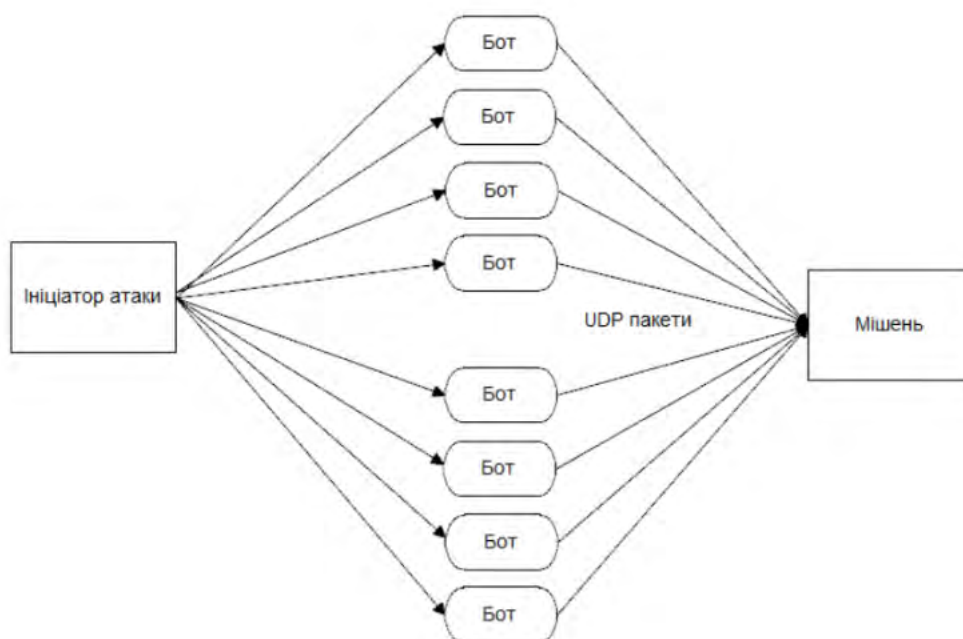


Рисунок 1.4 – Схема UDP флуду

ICMP флуд (англ. ICMP flood) – це різновид флуду як DoS атаки, в основі якого лежить генерування великої кількості ICMP пакетів, спрямованих на атакований вузол мережі. Internet Control Message Protocol (ICMP) – це протокол мережного рівня моделі OSI, який використовується для передачі системних мережних повідомлень та діагностики мережі [6]. Кожен раз на ICMP-запит має

йти відгук, тому відправка великої кількості запитів такого типу призводить до виснаження ресурсів атакованого мережного вузла, незважаючи на те що трафік є легітимним. Знову ж таки, акценти робляться на великій кількості запитів ніж на їхній певній структурованості або ж послідовності [4]. Схематичне зображення ICMP флуду концептуально виглядає так само, як на рис. 1.4.

IGMP флуд (англ. IGMP flood) – ще один різновид флуду, механізм якого є дуже подібним до ICMP. Internet Group Management Protocol (IGMP) – це сервісний протокол для управління мережними групами, подібний до ICMP за принципом роботи, який підтримує багатоадресну передачу, що збільшує негативний потенціал флуду для атакованого вузла мережі [4], [7].

1.2.2 Атаки, спрямовані на сервер

Атаки з цієї категорії мають на меті виснаження обчислювальних ресурсів серверу або його оперативної пам'яті, що потенційно може призвести до відмови в обслуговуванні. Ідея в тому, щоб використовувати потенційні вразливості серверу або мережних протоколів, аби змусити сервер постійно обробляти нелегітимні підключення та трафік, щоб в результаті не лишилося ресурсів на обробку запитів від легітимних користувачів. Основним напрямком таких атак є використання вразливостей стеку протоколів TCP/IP [4].

TCP SYN флуд (англ. TCP SYN flood) – це вид DoS атаки, спрямований на нелегітимне використання механізму трикратного рукостискання (англ. TCP handshake) протоколу TCP [4]. Transmission Control Protocol (TCP) – це протокол транспортного рівня моделі OSI, в основі якого лежить встановлення з'єднання між клієнтом та сервером перед початком передачі даних (англ. connection-oriented protocol) [8]. Механізм встановлення трикратного рукостискання відбувається за наступним сценарієм:

- клієнт відправляє SYN запит до сервера;

- сервер обробляє запит і відправляє у відповідь підтвердження та зустрічний запит на відкриття з'єднання SYN+ACK;
- клієнт відправляє підтвердження на відкриття з'єднання ACK;
- після цього з'єднання вважається встановленим і починається передача даних [8].

TCP SYN атака має на меті змусити сервер відкрити якомога більше потенційних з'єднань, не встановлюючи їх. Атакуюча робоча станція відправляє запит на встановлення TCP-з'єднання SYN до атакованого серверу, попередньо підміняючи свою справжню IP-адресу на випадкову. Сервер, зі своєї сторони, починає відкривати з'єднання та відправляє у відповідь підтвердження та зустрічний запит SYN+ACK. Оскільки IP-адреса клієнта була попередньо скомпрометована, запит від серверу буде відправлено випадковому мережному вузлу, тому відповідь ACK ніколи не буде відправлена до сервера. Наступні декілька секунд, або навіть, хвилин, сервер буде знаходитися в режимі очікування відповіді від клієнта та буде змушений тримати відкритим з'єднання, що потребує витрат ресурсів. Таким чином, для легітимних користувачів буде доступно на одне з'єднання менше. Так як у складі DDoS атаки бере участь велика кількість атакуючих робочих станцій, дуже швидко запас доступних активних з'єднань буде вичерпаний і легітимні користувачі втратять можливість встановити з'єднання з сервером що й призводить до відмови в обслуговуванні [4]. Нижче наведено схему атаки такого типу (рис. 1.5).

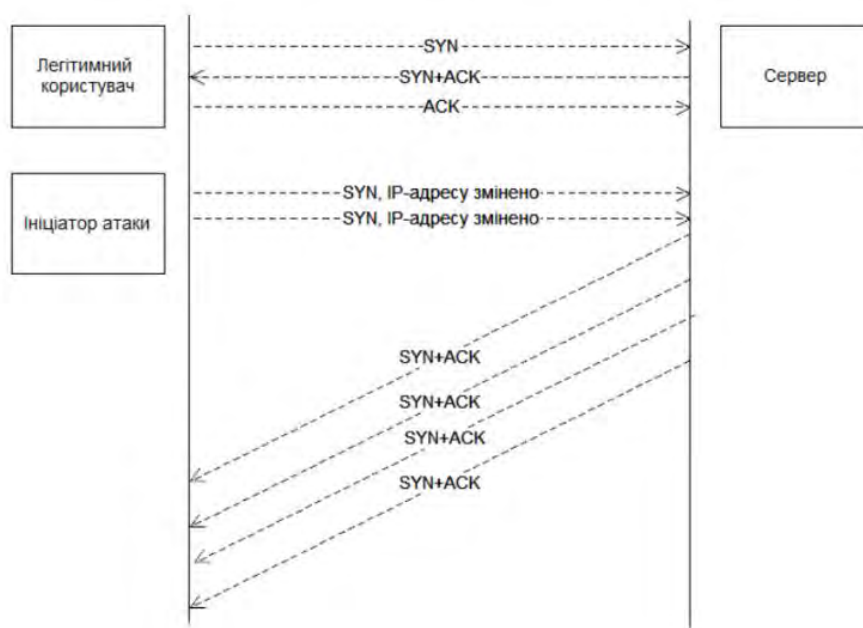


Рисунок 1.5 – Схема TCP SYN флуду

TCP RST атака спрямована на спричинення розриву легітимних з'єднань. TCP сегмент має в заголовку спеціальний RST флаг [8]. Якщо клієнт або сервер отримує сегмент, в якому даний флаг є встановленим, з'єднання буде миттєво розірване. Такий механізм у складі протоколу передбачено для тих випадків, коли подальша передача даних неможлива, або ж треба терміново розірвати з'єднання. TCP RST атака використовує цей механізм для створення умови відмови в обслуговуванні. Атакуюча робоча станція відправляє велику кількість RST запитів до цільового серверу, змінюючи IP-адреси та порядкові номери сегментів з метою включення такого запиту до вже створеного легітимного з'єднання між клієнтом та сервером. Коли це станеться, з'єднання буде розірвано. Так як у здійсненні DDoS атаки беруть участь тисячі робочих станцій, кожна з яких відправляє тисячі TCP RST пакетів на сервер, ймовірність вгадування порядкового номеру сегменту стає достатньо високою, тому легітимні з'єднання починають раптово закриватися, що призводить до відмови в обслуговуванні [4]. Нижче наведено схему атаки такого типу (рис. 1.6).

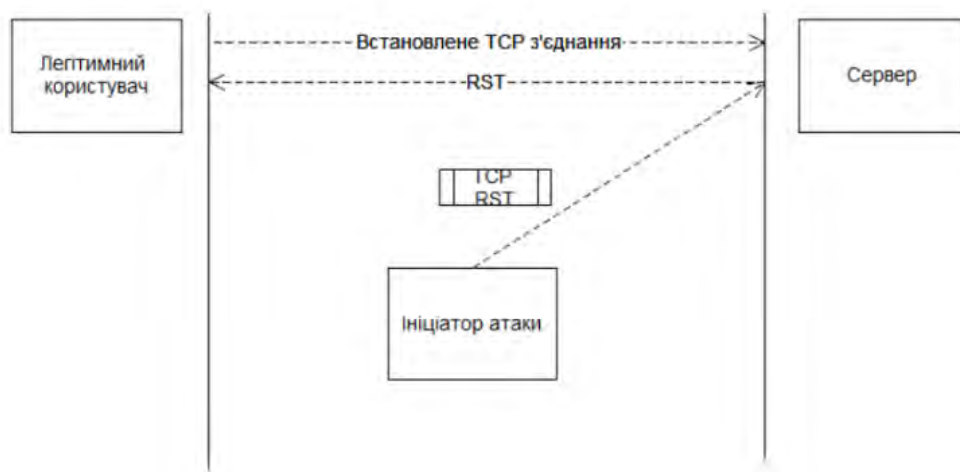


Рисунок 1.6 – Схема TCP RST атаки

TCP PSN ACK атака спрямована на виснаження ресурсів атакованого серверу шляхом відправлення великої кількості TCP запитів з встановленим флагом терміновості PSN. Коли сервер отримує TCP сегмент, в якому встановлено флаг PSN, це означає, що відповідь на запит необхідно відправити відразу, терміново [8]. Така операція потребує витрат системних ресурсів. За умови великої кількості запитів такого типу, ресурси серверу швидко вичерпуються, що унеможлиблює обслуговування легітимних клієнтів, внаслідок чого настає відмова в обслуговуванні [4]. Схема TCP PSN ACK атаки зображена на рис. 1.7.

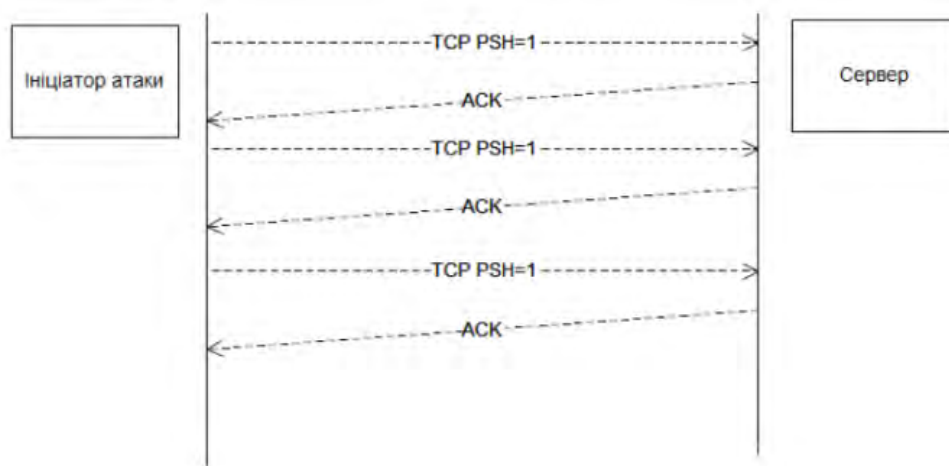


Рисунок 1.7 – Схема TCP PSN ACK атаки

1.2.3 Атаки, спрямовані на сервіс

Атаки з цієї категорії спрямовані на ураження сервісу як цілісного додатку, яким може виступати, наприклад, кінцевий вебсайт або хмарний сервіс. Основними векторами нападу є вразливості у мережних протоколах прикладного рівня моделі OSI та певні особливості їх роботи. Спектр протоколів достатньо широкий (HTTP, SMTP, FTP, DNS, VoIP), так само як і методів здійснення атаки (флуд, встановлення з'єднання, надмірне використання шифрування) [4].

HTTP флуд (англ. HTTP flood) – це дуже популярний й одночасно простий вектор атаки з категорії сервісних DoS атак, в основі якого лежить використання протоколу HTTP для здійснення легітимних запитів до сервісу. Таку атаку майже не можливо ідентифікувати виключно за змістом HTTP-повідомлень, адже вони можуть бути повністю правильними. Так як дана атака представляє собою флуд, в основі виникнення стану відмови в обслуговуванні лежить виснаження ресурсів сервісу через велику кількість запитів [4].

За таким самим принципом відбувається DNS флуд. Велика кількість правильних DNS запитів з робочих станцій ботнету відправляється на атакований DNS-сервер, який, опрацьовуючи таку кількість запитів одночасно, витрачає усі доступні ресурси, що призводить до виникнення умови відмови в обслуговуванні [4].

Більш складними є так звані «повільні» GET та POST HTTP-запити, в основі яких лежить максимальна фрагментація повідомлень. За нормальних умов, повідомлення вміщують максимально допустимий об'єм даних який можна передати за одне повідомлення. У разі «повільної» HTTP GET атаки, робоча станція робить декілька GET-запитів до атакованого сервісу, передаючи запит мінімальними об'ємами даних у кожному повідомленні періодично, з певним інтервалом часу, примушуючи сервіс тримати з'єднання відкритим тривалий час. Потім процес повторюється. За умови великої кількості робочих станцій у ботнеті, такі запити, з рештою, максимально займають ресурси системи, що призводить до

відмови в обслуговуванні легітимних користувачів [4]. Схема такої атаки зображена на рис. 1.8.

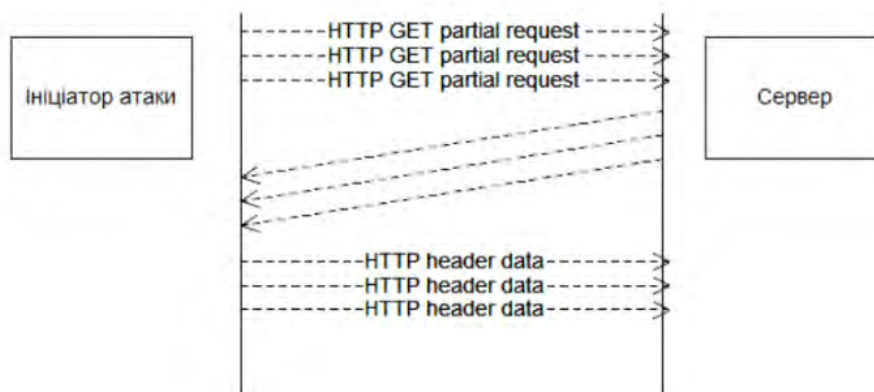


Рисунок 1.8 – Схема «повільної» HTTP GET атаки

Дуже подібною до попередньої є «повільна» HTTP POST атака, суть якої у відправленні великої кількості HTTP POST-запитів до сервісу розміром в один байт та через однакові інтервали часу. Сервіс, знаючи про довжину усього запиту, змушений чекати доки всі дані не будуть передані, оброблюючи запит дуже довго. За умови відкриття сотень таких «повільних» з'єднань, у сервісу не залишається ресурсів для обробки легітимних користувачів, що з рештою призводить до відмови в обслуговуванні [4]. Схема такої атаки зображена на рис. 1.9.

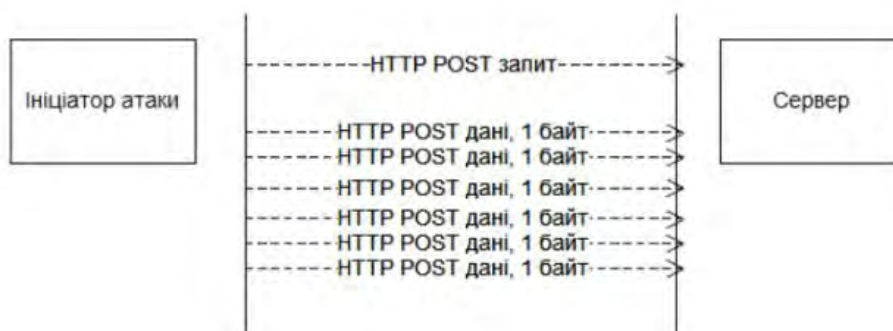


Рисунок 1.9 – Схема «повільної» HTTP POST атаки

1.3 Методи протидії DDoS атакам

Преш за все варто пам'ятати, що не одна система не є повністю безпечною, тільки безпечнішою. Взагалі, атаки типу DDoS, є найскладнішими в плані захисту від них, адже є дуже багато векторів потенційного розвитку атаки. Подекуди все виглядає просто як великий обсяг легітимного трафіку, тому встановити точний факт атаки буває досить складно. Саме тому протидія атакам типу DDoS має бути комплексною.

По-перше, завчасно організована стратегія захисту та план протидії будуть корисними, адже під час самої атаки важлива кожна хвилина.

По-друге, має бути завчасно проведена оцінка потенційних наслідків та закладено допустимий бюджет на протидію атаці, адже навіть якщо атака буде зупинена швидко (на протязі першої години від початку це доволі непоганий результат), певних збитків компанія та користувачі все одно зазнають.

По-третє, команда з кібер-безпеки має бути постійно готова та мати чіткий план протидії, адже не можливо передбачити коли саме розпочнеться атака.

Також, дуже важливо проводити збір даних стосовно кожної атаки, щоб виявляти слабкі місця в захисті і збирати легітимний трафік, аби створити базу даних легітимних IP-адрес яка пізніше може бути використана для фільтрації легітимного трафіку під час DDoS атаки.

Наступним кроком є встановлення системи виявлення та захисту від DDoS атак. Така система має бути комплексною і є найважливішим компонентом захисту, адже своєчасне виявлення факту атаки може зменшити її вплив шляхом своєчасної протидії (наприклад, увімкненням фільтрації по базі даних IP-адрес легітимних користувачів). Розробка такої системи і є одним з результатів дослідження, проведених в рамках цієї роботи [4].

Останнім пунктом є симуляція DDoS атак задля перевірки захищеності системи, виявлення слабких місць у захисті, підготовки команди.

1.4 Огляд засобів для здійснення DDoS атак

Існує великий спектр програмного забезпечення що генерує DDoS трафік. Воно може мати форму як шкідливого ПЗ, що розповсюджується проти волі користувачів, робочі станції котрих стають учасниками ботнету без відому власне користувачів, так і вільного ПЗ, яке користувачі встановлюють добровільно щоб взяти участь у DDoS атаці. Другий спосіб організації ботнету називається волонтерським.

У рамках даної роботи було досліджено два найпопулярніші представники «вільного» ПЗ для здійснення атак типу DDoS – це LOIC (англ. Low Orbit Ion Cannon) та hping3.

LOIC – це безкоштовне програмне забезпечення з відкритим вихідним кодом, розроблено на мові програмування C# (англ. CSharp), яке має зручний та простий графічний інтерфейс користувача, метою якого, перш за все, було тестування мережі під високим навантаженням. Пізніше воно було модифіковане для можливості віддаленого управління, що дозволило його використання у складі керованого ботнету. Головна функція LOIC зводиться до генерування UDP, TCP або HTTP флуду в напрямку IP-адреси або доменного імені цільового ресурсу. Все що треба зробити, це вказати ціль, протокол, кількість одночасних потоків виконання та швидкість генерування трафіку і почати флуд. Єдиним недоліком LOIC є неможливість маскуванню IP-адреси атакуючого, що може призвести до виявлення особи нападника та призвести до юридичних наслідків [4], [9]. Скріншот головного вікна LOIC зображено на рис. 1.10.

На основі дослідження трафіку, згенерованого програмним забезпеченням LOIC, був розроблений алгоритм протидії UDP флуду, що буде викладений в наступних розділах роботи.



Рисунок 1.10 – Головне вікно програмного забезпечення LOIC

hping3 – це програмне забезпечення з відкритим вихідним кодом, написане на мові програмування C, яке також було розроблено з метою тестування мережі шляхом генерування трафіку, проте може успішно використовуватися як інструмент створення DDoS атаки. За його допомогою можна генерувати TCP та ICMP трафік з різною швидкістю, направленою на вказану IP-адресу. На відміну від LOIC, дане програмне забезпечення дає змогу змінювати IP-адресу генерованих пакетів на випадкову, що ускладнює ідентифікацію робочої станції яка здійснює атаку. Тому hping3 є частим вибором організаторів DDoS атак. hping3 не має графічного інтерфейсу користувача, тільки інтерфейс командної строки [4], [10]. Скріншот прикладу команди для запуску зображений на рис. 1.11.

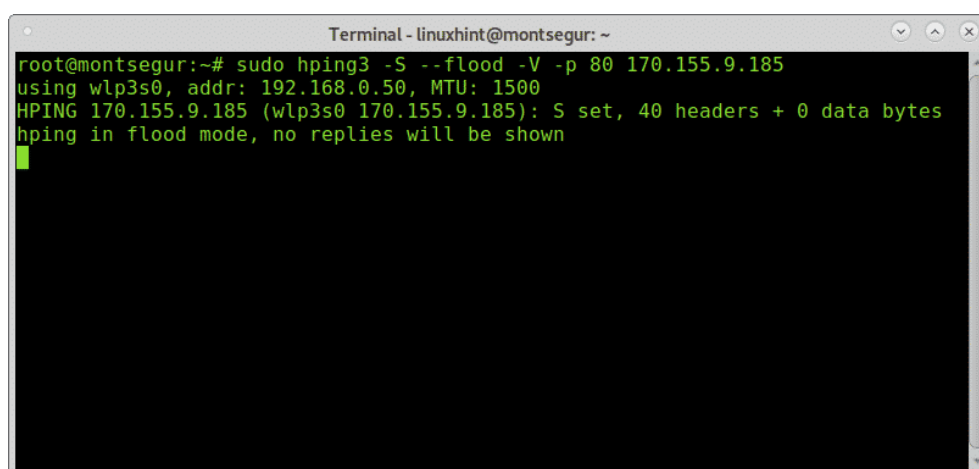


Рисунок 1.11 – Команда запуску TCP-флуду за допомогою hping3

На основі дослідження трафіку, згенерованого програмним забезпеченням hping3, було досліджено та реалізовано алгоритм протидії DDoS атакам типу флуд на основі аналізу IP-адрес, який викладений у наступному розділі.

1.5 Огляд алгоритму виявлення атак типу DDoS на основі розрахунку ентропії Шеннона

У цьому розділі описується досліджений алгоритм виявлення присутності атак типу DDoS на основі розрахунку ентропії Шеннона, авторства Liying Li, Jianying Zhou, та Ning Xiao з національного університету Сінгапуру (англ. National University of Singapore) [11].

Перш за все варто загадати, що існує два основні методи виявлення комп'ютерних атак – виявлення, що базується на перевірці електронного підпису (мітки) та виявлення аномалій.

Перевірка електронного підпису є основним методом виявлення шкідливого ПЗ або трафіку через свою простоту та надійність. Будь-яка інформація може мати власний унікальний підпис, тому атаку можливо ідентифікувати максимально точно простим зіставленням з підписом, який є відомим завчасно. Проте такий метод повністю безнадійний у разі виникнення нової, ще невідомої вразливості. Навіть якщо новий вектор атаки було точно ідентифіковано, оновлення правил перевірки займає час, за який атака продовжується не зустрічаючи будь-якого опору. Також, перевірка підпису не може бути використана для ідентифікації флуду, адже генерований трафік може бути цілком легітимним в своїй основі.

Виявлення аномалій є більш гнучким способом ідентифікації атак, адже не потребує точних параметрів для зіставлення зі зразком. В основі цього методу лежать поняття нормальної та аномальної поведінки, а також певна умова, за якої система переходить з нормального стану в аномальний. Як правило, такою умовою є перевищення завчасно встановленого порогового значення (англ. threshold,

baseline), яке може бути налаштоване вручну або базуючись на певних дослідженнях та розрахунках. У разі виникнення аномалії, система має видати відповідне повідомлення. Використовуючи метод виявлення аномалій є можливою якісна ідентифікація флуду, навіть якщо трафік є легітимним.

Важливе місце у виявленні аномалій займає розрахунок ентропії Шеннона. Ентропія Шеннона, або інформаційна ентропія (англ. Shannon Entropy, information entropy) – це міра невизначеності, пов'язана з випадковою (статистичною) величиною. Вона може бути інтерпретована як середня найменша довжина повідомлення, у бітах, яке може передати корисну інформацію. Це визначає фундаментальний математичний ліміт стиснення даних без втрат. Концепт інформаційної ентропії був представлений Клодом Шенноном (англ. Claude E. Shannon) у 1948 році й викладений в роботі «Математична теорія комунікації» (англ. «A Mathematical Theory of Communication»).

Інформаційна ентропія може бути використана для розрахунку випадковості розподілу певних атрибутів мережних пакетів, таких як вихідна IP-адреса. Наприклад, аналізатор отримав 1000 послідовних мережних пакетів на певний момент часу, та розраховує розподіл IP-адрес цих пакетів на інтервалі в 1000 пакетів. Подальший розрахунок ентропії отриманого розподілу може дати відповідь на запитання стосовно випадковості вихідної послідовності пакетів. Оскільки природа DDoS атак така, що під час неї, мережний трафік нараховує велику кількість пакетів від великої кількості IP-адрес (тому що трафік під час атаки генерується ботнетом з великої кількості робочих станцій), загальний мережний трафік буде мати розподіл IP-адрес великого ступеня випадковості, тому ентропія буде мати високе абсолютне значення. Знаючи цей факт, можливо зробити імплементацію алгоритму виявлення DDoS атак шляхом розрахунку ентропії розподілу IP-адрес у вихідному трафіку на певних пакетних інтервалах. Чим більше значення ентропії, тим вищим є ступінь випадковості мережного трафіку, що є індикатором початку DDoS атаки.

Формула (1.1) розрахунку ентропії Шеннона має наступний вигляд:

$$H = - \sum_{i=1}^n p_i \log_2 p_i, \quad (1.1)$$

де p_i – ймовірність появи унікальної IP-адреси;

n – кількість унікальних IP-адрес.

Можливо встановити порогове значення для ентропії T , та виконати перевірку за формулою (1.2):

$$A = \begin{cases} 1, & H > T \\ 0, & H < T \end{cases} \quad (1.2)$$

де A – логічне значення присутності атаки.

Проте такий спосіб розрахунку призведе до помилкового виявлення аномалій мережного трафіку, адже раптовий ріст кількості унікальних IP-адрес може бути результатом збільшення активності легітимних користувачів. Підозра стосовно аномалії має бути підтверджена на певній послідовній кількості мережного трафіку задля уникнення помилкових результатів. Існує декілька варіантів реалізації таких обчислень, проте зупинимось на методі розрахунку ентропії на часових інтервалах.

Введемо поняття інтервалу часу ΔT , та порогове значення ентропії ω . Побудуємо вектор критичних значень X^T , з фіксованим розміром в N елементів, які можуть приймати значення в діапазоні $[-1, 1]$. Початково вектор ініціалізований значеннями -1 (1.3):

$$X^T = [-1, -1, \dots, -1, -1]. \quad (1.3)$$

Введемо поняття порядкового номеру ітерації j . Кожна ітерація відбувається через рівні інтервали часу ΔT , та відповідає певному елементу з вектору критичних значень X^T . Ітерації починаються з моменту старту аналізу

мережного трафіку з початкового індексу вектору. На кожній ітерації корегується значення відповідного індексу вектору за формулою (1.4):

$$X_j^T = \begin{cases} +1, & H_j \geq \omega \\ -1, & H_j < \omega \end{cases} \quad (1.4)$$

де H_j – розраховане значення ентропії на ітерації.

Після проходження N ітерацій, тобто після завершення одного повного проходу вектору критичних значень, процес повторюється циклічно на протязі всього часу аналізу мережного трафіку, постійно корегуючи вектор критичних значень.

Умовою виявлення присутності аномалії, а отже й DDoS атаки, є переведення всіх значень вектору X^T у значення +1 (1.5):

$$X^T = [+1, +1, \dots, +1, +1]. \quad (1.5)$$

Перевірка вектору критичних значень здійснюється після проходження усіх N ітерацій, що відповідають його індексам. Таким чином, для ідентифікації аномалії мережного трафіку, необхідно спостерігати її на протязі певного мінімального інтервалу часу τ , який можна розрахувати за формулою (1.6):

$$\tau = \Delta T \cdot N. \quad (1.6)$$

Параметри ΔT та N можна корегувати для налаштування чутливості системи виявлення. Розрахунок ентропії зупиняється у разі відсутності мережних пакетів за останній інтервал часу ΔT .

1.6 Існуючі програмні засоби що вирішують аналогічні завдання

CloudFlare – це продукт однойменної компанії дистрибуції хмарних сервісів, що є лідером в області комерційних систем захисту від атак типу DDoS. Займає більшу частину ринку подібних рішень, пройшов атестацію від світових лідерів. Дуже часто можна спостерігати перенаправлення на сервіс DDoS захисту CloudFlare на веб-сайтах великих та середніх за розмірами компаній.

Azure DDoS Protection – це хмарне рішення для захисту від DDoS атак, розроблене корпорацією Microsoft. Має дуже великий потенціал інтеграції з вже розгорнутими рішенням у Microsoft Azure Cloud, але майже не використовується як окремий продукт.

Kaspersky DDoS Protection – комплексне рішення захисту від атак типу DDoS, яке пропонує компанія Kaspersky Lab. Відмінність від інших рішень полягає в тому, що система може бути розгорнута як на основі хмарних технологій, так й у вигляді фізичного пристрою встановленого в дата центрі.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Дослідження алгоритму виявлення аномалій мережного трафіку на основі розрахунку кумулятивної суми

У розділі 1.5 було розглянуто алгоритм виявлення DDoS атак на основі виявлення аномалій мережного трафіку методом розрахунку інформаційної ентропії (ентропії Шеннона).

У повному докладі стосовно цього алгоритму було розглянуто ще один метод динамічного аналізу мережного трафіку на присутність аномалій – «метод кумулятивної суми» (англ. CUSUM method). Він базується на постійному прирості кумулятивної ентропії до моменту досягнення порогового значення, що й визначає факт присутності аномалії [11].

Нехай X_n – значення ентропії, розраховане на кількості пакетів за проміжок часу Δ_n , та $\{X_n\}$ – послідовність проаналізованих пакетів. За звичайних умов, розподіл IP-адрес у послідовності є нормальним. Припустимо, що параметром варіативності є середнє значення послідовності $\{X_n\}$. Перед зміною стану, це значення $E(X_n) = \alpha$, є дуже малим та завжди позитивним. Перед атакою, коли мережний трафік знаходиться у нормі, розподіл IP-адрес є стабільним й має малу випадковість, тому значення ентропії залишається малим. Проте під час DDoS атаки, середнє значення стрімко зростає, і значення $E(X_n)$ стає постійно більшим ніж α .

Метод кумулятивної суми базується на припущенні, що за нормальних умов, середнє значення випадковості послідовності IP-пакетів буде негативним, тоді як позитивним – після зміни поведінки мережного трафіку. Введемо поняття негативного значення послідовності $\{Z_n\} = X_n - \beta$, де $\alpha = \alpha - \beta$. Значення β є константою для генерації негативного значення ентропії. Автори використовують значення $\beta = 2\alpha$, маючи на увазі, що коли значення ентропії буде в два рази вищим за порогове значення, можна стверджувати про появу аномальної активності в мережі та починати підсумовування ентропії. Тобто, під час DDoS атаки, Z_n

раптово виросте до великого позитивного значення. Збільшення кумулятивної суми, в решті решт, призведе до перевищення її порогового значення що й буде фактом виявлення атаки.

Рекурсивна формула для кумулятивної суми має наступний вигляд (2.1):

$$\begin{cases} y_n = (y_{n-1} + Z_n)^+ \\ y_0 = 0 \end{cases}, \quad (2.1)$$

де $x^+ = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$, й представляє собою кумулятивне значення Z_n .

Чим більше y_n , тим потужніша атака. За проміжок часу аналізу τ_N , функція визначення факту присутності атаки має вигляд (2.2):

$$d_N(y_n) = \begin{cases} 0, & (y_n \leq N) \\ 1, & (y_n > N) \end{cases}, \quad (2.2)$$

де $d_N(y_n)$ – це вердикт за проміжок часу n ;

N – порогове значення.

Після дослідження цього методу був зроблений висновок, що він не має механізму зворотного зменшення кумулятивної суми, що не підходить для розробки системи аналізу мережного трафіку на великих проміжках часу. Причина в тому, що періодичні та короткочасні збільшення активності у мережі, будуть призводити до підсумовування кумулятивної суми кожного разу, що, через певний час, призведе до перевищення порогового значення кумулятивної суми. Оскільки тимчасові підвищення активності на великих проміжках часу не можуть бути інтерпретовані як DDoS атака, використання цього методу може призвести до хибних висновків стосовно її присутності. Отже, після дослідження даного методу було прийняте рішення використовувати метод підрахунку та аналізу значення ентропії на часових інтервалах, описаного у розділі 1.5.

2.2 Дослідження алгоритму виявлення аномалій мережного трафіку на основі розрахунку інформаційної ентропії на часових інтервалах

Після дослідження опису методу аналізу мережного трафіку шляхом розрахунку інформаційної ентропії на часових інтервалах [11], був зроблений висновок стосовно гнучкості та ефективності цього методу та розроблений теоретичний план його реалізації.

Для реалізації алгоритму знадобиться два потоки виконання. Перший – потік збору IP-пакетів з мережі (надалі під назвою PacketThread). Другий – потік аналізу послідовностей IP-адрес, створених першим потоком (надалі під назвою EvaluationThread).

PacketThread виконується постійно, з моменту запуску програми, та здійснює забір усіх IP-пакетів з мережного трафіку, зчитує IP-адресу, зберігає її, та одночасно підраховує загальну кількість IP-пакетів. Для збереження зчитаних IP-адрес з пакетів підходить така структура даних як хеш-таблиця (словник). Оскільки необхідно зберігати унікальні IP-адреси, можна створити хеш-таблицю з ключами у вигляді IP-адрес, та значеннями у вигляді кількості зареєстрованих пакетів, що мають IP-адресу в якості ключа. Хеш-таблиця, як структура даних, має таку властивість, що ключі повинні бути унікальними. Також важливим критерієм вибору структури даних для збереження інформації для аналізу є швидкодія операцій з нею. Хеш-таблиця, у більшості випадків, має константну складність на пошук, вставку та видалення елементів. Саме тому така структура даних як хеш-таблиця (словник) була обрана для збереження унікальних IP-адрес та кількості їх появи [12].

Структурну схему організації збереження даних для аналізу наведено на рис. 2.1.

Кожного разу коли в мережі з'являється новий пакет, він має бути зчитаним у потоці PacketThread. Проте суттєвими є тільки IP-пакети, тому всі інші мають фільтруватися.

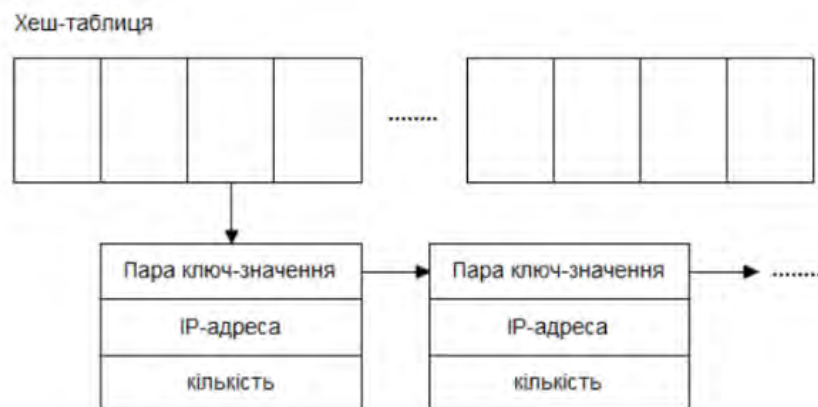


Рисунок 2.1 – Схема організації структури даних для збереження IP-адрес

Блок-схему роботи потоку зображено на рис. 2.2.

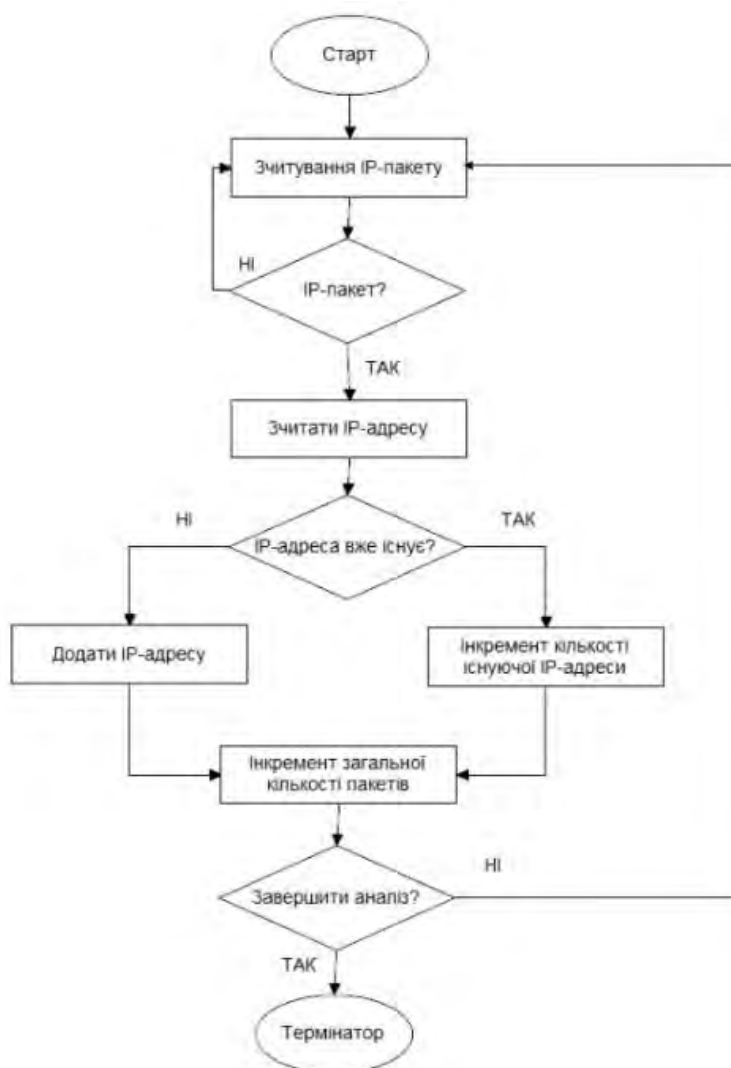


Рисунок 2.2 – Блок-схема логіки потоку зчитування пакетів PacketThread

Потік аналізу IP-адрес EvaluationThread здійснює розрахунок інформаційної ентропії, перевірку на перевищення порогового значення, оновлення критичного значення за поточним індексом та перевірку вектору критичних значень після його повного поіндексного проходу. Потік виконується постійно, з моменту початку аналізу, та запускається з потоку пакетів PacketThread. Оновлення вектору критичних значень здійснюється поіндексно, з встановленим інтервалом часу, на основі розрахунку інформаційної ентропії та перевірки перевищення її порогового значення. Після проходу вектору по всій довжині, здійснюється перевірка його значень, і у разі виявлення аномалії, виводиться відповідне повідомлення. Кожного разу після оновлення значення у векторі, загальна кількість пакетів анулюється, а хеш-таблиця IP-адрес – очищується цим же потоком.

В якості вектору критичних значень можна використовувати структуру даних «вектор» фіксованої довжини, а для вимірювання інтервалів часу підходить звичайний таймер.

Блок-схема роботи потоку зображена на рис. 2.3.

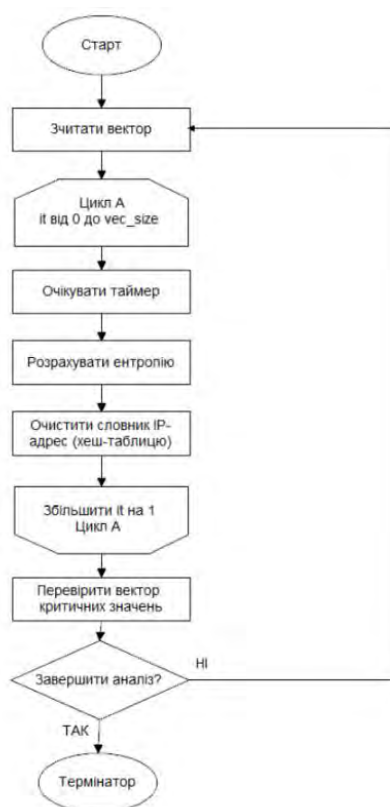


Рисунок 2.3 – Блок-схема логіки потоку аналізу EvaluationThread

Синхронізована робота двох вищезгаданих потоків і є вдосконаленою реалізацією алгоритму виявлення аномалій мережного трафіку на основі розрахунку інформаційної ентропії на інтервалах часу. Проте така реалізація має певні зміни у порівнянні з описом у першоджерелі.

По-перше – не використовується поняття «пакетного вікна» [11]. Достатньо аналізу IP-адрес пакетів, зчитаних на часових інтервалах. У власній реалізації використовуються сталі інтервали часу, на відміну від сталого розміру «пакетного вікна». Таким чином, отримуємо різну кількість IP-адрес для аналізу на кожному проміжку часу. Але оскільки розрахунок інформаційної ентропії базується на розподілі IP-адрес, результат буде сталим. Наприклад, маємо 10 пакетів з двох IP-адрес, по 5 на кожен. Тоді маємо ймовірність появи кожної з двох адрес 0.5. Якщо пакетів буде більше, скажімо, 1000, але розподіл не зміниться (дві унікальні IP-адреси, кожній відповідає по 500 пакетів), то ймовірність залишається сталою – 0.5. Таким чином, не використовуючи «пакетне вікно», алгоритм можливо спростити.

По-друге – використовується чиста хеш-таблиця з парами <IP-адреса, кількість>, що також спрощує алгоритм, адже у першоджерелі було запропоновано використання окремої структури для збереження пар, що має під собою додаткову ініціалізацію об'єктів цієї структури [12].

На основі дослідження DDoS трафіку, згенерованого спеціалізованим програмним забезпеченням hping3, а також аналізу легітимного трафіку, взятого з публічного доступу у вигляді зразків, було теоретично розраховано параметри алгоритму: порогове значення інформаційної ентропії – 2.0, інтервал часу – 2 секунди, розмір вектору критичних значень – 15 елементів. Тобто, для здійснення висновку стосовно присутності DDoS атаки, необхідно щоб інформаційна ентропія перевищувала порогове значення на протязі 30 секунд. Проте має бути можливість для налаштування даних параметрів у системі за допомогою конфігурації.

2.3 Дослідження та розробка алгоритму виявлення UDP флуду

UDP (англ. User Datagram Protocol) – це протокол транспортного рівня моделі OSI, який використовується для швидкої передачі даних між клієнтом та сервером без встановлення з'єднання, забезпечення цілісності даних й порядку слідування пакетів [5]. Це альтернатива протоколу TCP (англ. Transmission Control Protocol), який забезпечує цілісність та порядок слідування пакетів через встановлення з'єднання між клієнтом та сервером [8]. Через простоту та швидкодію, UDP використовується в таких мережних сервісах, для яких швидкість передачі даних є важливішою навіть якщо певна кількість пакетів буде втрачена. Наприклад, аудіо та відео трансляції, віддалене керування процесами, служба DNS, тощо. Структура UDP пакету значно простіша у порівнянні з TCP, що робить процес генерування трафіку швидким та «легким» з точки зору витрат обчислювальних ресурсів. Саме тому UDP флуд – це розповсюджений вектор DDoS атак, тому проблема його ідентифікації та блокування є актуальною.

Існує певна кількість протоколів прикладного рівня що використовують UDP. Наприклад, RTP, RTMP чи DNS. Головна проблема розробки алгоритму виявлення UDP флуду – це ідентифікація його серед іншого легітимного UDP трафіку, адже в своїй основі, флуд складається з правильно сформованих UDP пакетів. Тому необхідно проводити більш глибокий аналіз послідовності пакетів, ніж проста перевірка на правильність формування пакету, з урахуванням того факту, що це може бути й звичайний, легітимний, UDP трафік. Як відрізнити UDP флуд від цілком допустимої аудіо- чи відео-трансляції з використанням протоколу RTP або RTMP? Щоб відповісти на це запитання, варто спочатку розглянути структуру заголовка UDP пакету, яка зображена на рис. 2.4.

Біти	0-15	16-31
0-31	Порт відправника (src port)	Порт одержувача (dst port)
32-63	Довжина пакету (len)	Контрольна сума (checksum)

Рисунок 2.4 – Структура заголовка UDP пакету

Структура заголовку UDP дуже проста й складається лише з чотирьох компонентів:

- порт відправника (src port) – порт клієнта;
- порт одержувача (dst port) – порт сервера;
- довжина пакету (len) – довжина всього UDP пакету разом з даними;
- контрольна сума (checksum) – хеш-значення даних пакету [5].

Розроблений алгоритм аналізує контрольні суми послідовностей UDP пакетів в рамках одного мережного потоку. Мережний потік (англ. flow) – це послідовність мережних пакетів, що мають однакові порти/адреси відправника та одержувача. Тобто, всі пакети в межах одного потоку є логічно пов'язаними між собою, тому аналіз тільки таких послідовностей має сенс.

Контрольна сума – це результат хешування даних пакету, отриманий в результаті обчислення хеш-функції для цих даних – значення хешу [5]. Отже, для двох пакетів, що мають однакові дані, контрольні суми будуть однаковими. Алгоритм базується на припущенні, що пакети у складі UDP флуду матимуть однакові дані в кожному пакеті, тому їх контрольні суми будуть однакові на протязі всього UDP-потoku. Проаналізувавши певну кількість послідовних UDP-пакетів у рамках одного потоку на однаковість їх контрольних сум, що неможливо у випадку легітимного трафіку, можна зробити висновок про присутність UDP флуду. Дане припущення базується на результатах дослідження зразків RTP трафіку та DDoS трафіку (а саме UDP флуду), згенерованого програмним забезпеченням LOIC.

Розглянемо зразок звичайної RTP-сесії онлайн трансляції. Її візуалізацію було отримано з використанням програмного забезпечення для аналізу мережного трафіку Wireshark та зображено на рис. 2.5.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	193.63.53.155	224.1.2.3	UDP	78	1177 → 6003 Len=36
2	0.000072	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
3	0.000112	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
4	0.000151	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
5	0.000190	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
6	0.000230	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
7	0.000268	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
8	0.000307	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
9	0.000346	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
10	0.000383	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
11	0.000420	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116
12	0.000448	193.63.53.155	224.1.2.3	UDP	1158	1177 → 6003 Len=1116

> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
 > Ethernet II, Src: Dell_e5:f5:52 (00:0f:1f:e5:f5:52), Dst: IPv4mcast_01:02:03 (01:00:5e:01:02:03)
 > Internet Protocol Version 4, Src: 193.63.53.155, Dst: 224.1.2.3
 > User Datagram Protocol, Src Port: 1177, Dst Port: 6003
 Source Port: 1177
 Destination Port: 6003
 Length: 44
 Checksum: 0x8810 [unverified]

```

0000  01 00 5e 01 02 03 00 0f 1f e5 f5 52 08 00 45 00  --R--E-
0010  00 40 5e 9e 00 00 ff 11 84 2f c1 3f 35 9b e0 01  --^--.-./-?5-
0020  02 03 04 99 17 73 00 2c 88 10 13 07 de 87 c1 3f  --S-,-----?
0030  32 cf 4e cf 59 44 04 00 00 c1 43 42 53 99 00 09  --2-N-YD-.-CBS-
0040  cc ca 80 00 88 06 c1 3f 32 cf 05 5b 8b 07       -------?2-[-
  
```

Рисунок 2.5 – Візуалізація RTP-сесії онлайн трансляції

Можна побачити, що RTP-сесія складається з послідовності UDP-пакетів, кожен з яких містить різні дані всередині, а отже – різні значення контрольних сум, що також видно під час аналізу (значення у полі Checksum завжди різне). Звичайно, можливі дублікати пакетів навіть у разі легітимного UDP-трафіку, проте вони навряд чи будуть послідовними. В іншому випадку – довжина таких послідовностей буде незначною.

Тепер розглянемо UDP-флуд, згенерований програмним забезпеченням для здійснення DDoS атак LOIC. Його візуалізація зображена на рис. 2.6.

Можна побачити, що в рамках окремого мережного потоку, UDP-пакети містять однакові дані, а отже – однакові значення контрольних сум на всій послідовності пакетів. Таким чином, можна встановити порогове значення кількості послідовних UDP-пакетів з однаковими значеннями контрольних сум, при досягненні якого можна зробити висновок щодо присутності UDP флуду як різновиду DDoS атаки. Звісно, атакуюча робоча станція може використовувати складніше ПЗ, яке буде змінювати дані пакетів випадково і кожного разу, або через

певні інтервали часу. Але частіше зломисники йдуть найпростішим шляхом, тому більшість випадків UDP флуду буде виявлена розробленим алгоритмом.

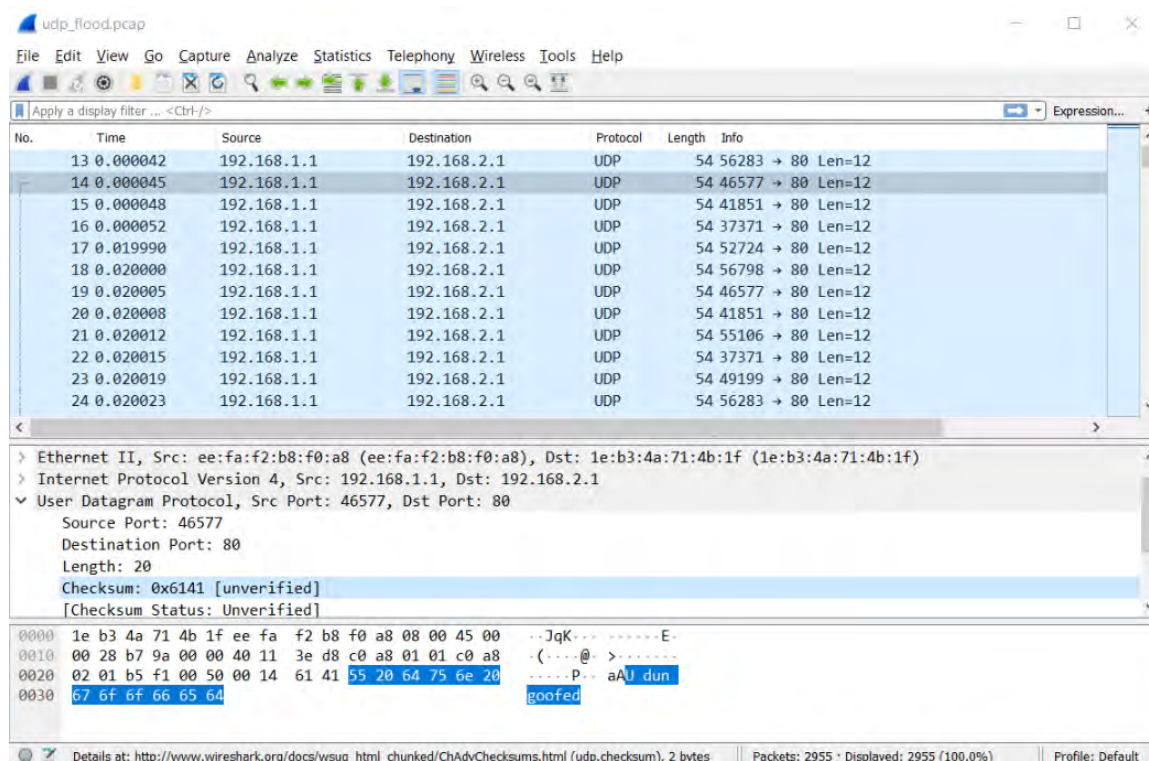


Рисунок 2.6 – Візуалізація UDP-флуду, згенерованого LOIC

Реалізація алгоритму базується на паралельному використанні двох потоків виконання. Перший – це потік зчитування UDP-пакетів (надалі PacketThread), а другий – потік аналізу послідовностей UDP-пакетів по кожному мережному потоку (надалі EvaluationThread).

Потік зчитування UDP-пакетів має на меті постійне зчитування пакетів з мережі, їх фільтрацію за протоколом UDP, перевірку на необхідність блокування та додавання нових пакетів у словник (хеш-таблицю) контрольних сум та словник (хеш-таблицю) UDP-потоків. Окремо існує словник (хеш-таблиця) заблокованих UDP-потоків, яка перевіряється потоком зчитування що блокує пакети у разі присутності останніх в словнику [12].

Блок-схему потоку зчитування UDP-пакетів наведено на рис. 2.7.

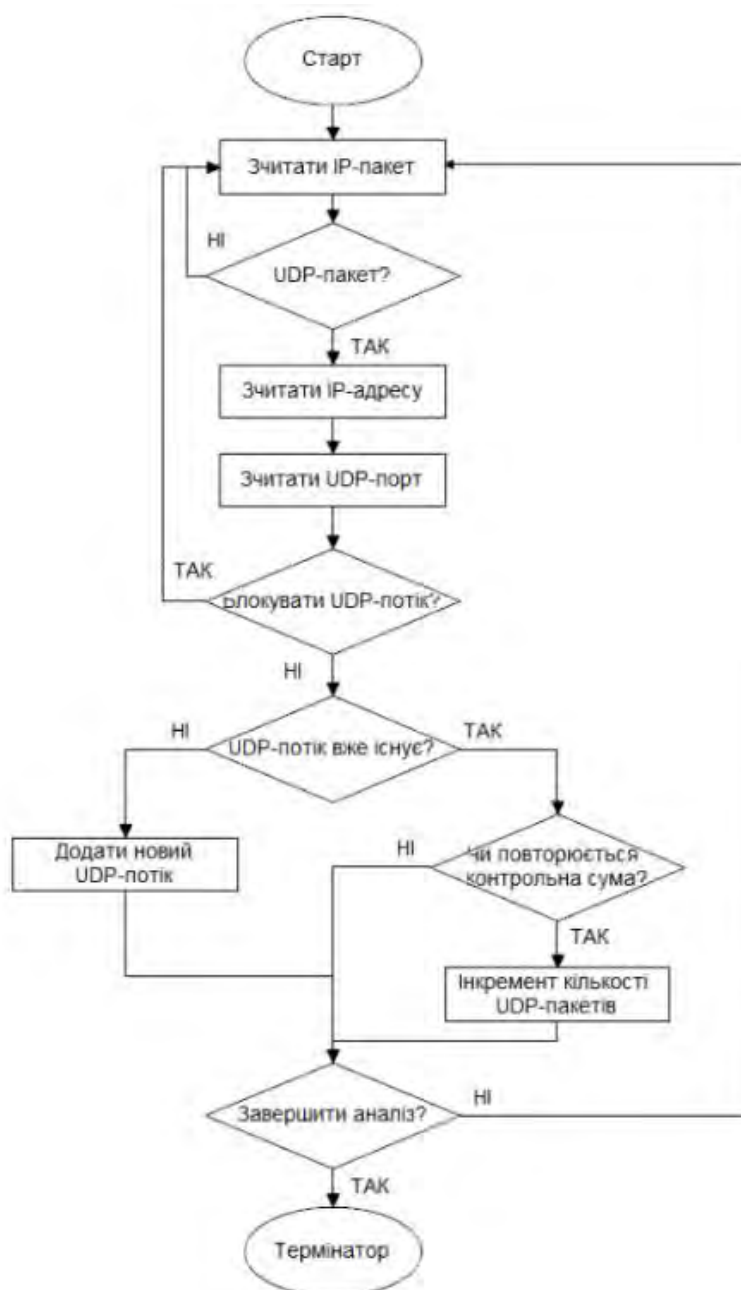


Рисунок 2.7 – Блок-схема роботи потоку зчитування UDP-пакетів PacketThread

Потік аналізу послідовностей UDP-пакетів по кожному мережному потоку EvaluationThread має на меті здійснення перевірки на присутність послідовностей пакетів з однаковими значеннями контрольних сум, та у разі виявлення таких – виводити відповідне повідомлення. Умовою перевірки присутності аномалії мережного трафіку є перевищення довжини послідовності пакетів у потоці сталого порогового значення. Також має бути передбачена можливість або тільки виводу повідомлення, або виводу повідомлення і додавання UDP-потіку до таблиці

заблокованих потоків, по якій здійснюється фільтрація UDP флуду в пакетному потоці PacketThread. Потік виконується періодично, через сталі проміжки часу, за який здійснюється забір потоків для аналізу, та до закінчення аналізу пакетів користувачем. Запускається з пакетного потоку PacketThread. Після здійснення аналізу, таблиці потоків та контрольних сум очищуються. Виконання потоку через встановлені інтервали часу може бути реалізоване звичайним таймером.

Блок-схему потоку аналізу послідовностей UDP-пакетів зображено на рис. 2.8.



Рисунок 2.8 – Блок-схема потоку аналізу послідовностей UDP-пакетів
EvaluationThread

Синхронізована робота двох вищезгаданих потоків і є теоретичною реалізацією розробленого алгоритму виявлення UDP флуду як різновиду DDoS атаки, що базується на аналізі контрольних сум послідовних пакетів у UDP-потоках.

На основі додаткового аналізу зразків легітимного UDP трафіку та UDP флуду, було теоретично розраховано оптимальні значення параметрів алгоритму, а

саме: інтервал часу аналізу – 2 секунди, порогове значення кількості послідовних UDP-пакетів з однаковими значеннями контрольних сум у складі одного потоку – 30 пакетів. Проте має бути можливість для налаштування даних параметрів у системі за допомогою конфігурації.

2.4 Дослідження та розробка схеми роботи системи як цілісного програмного забезпечення

Розроблювана система представляє собою програмне забезпечення для аналізу і фільтрації мережного трафіку, а отже, загальна схема має включати комбінацію логіки зчитування і передачі мережного трафіку та його аналізу. Можна виділити таку логічну структуру як «пакет». Мета програми – передати пакет з вхідного інтерфейсу на вихідний з накладанням логіки аналізу та, за необхідності, блокувати нелегітимні пакети.

Узагальнену схему логіки роботи системи зображено на рис. 2.9.



Рисунок 2.9 – Узагальнена схема логіки роботи системи

Після проведення дослідження можливих варіантів схеми системи, було обрано схему ланцюга відповідальності [13]. Так як маємо певну структуру «пакет», є доцільним передавати його від моменту зчитування з мережі, через всі компоненти аналізу, до передачі у вихідний інтерфейс або блокування. У разі блокування, пакет далі не передається. Таким чином, клієнт буде захищений та/або проінформований про присутність атаки. Так як пакети проходять через компоненти аналізу послідовно, один і той же пакет буде проаналізовано кожним методом. Ланцюг відповідальності виконується постійно, з моменту старту, доки користувач не зупинить процес.

3 ОПИС ПРОГРАМИ

3.1 Структура та функціонал розроблюваної програми

Розроблюване програмне забезпечення представляє собою потоковий аналізатор мережного трафіку на присутність атак типу DDoS. Загальна структура нагадує структуру фаєрволу або IPS/IDS (система виявлення та протидії мережним атакам), з орієнтацією на особливості предметної області.

Схема роботи базується на такому понятті як мережний інтерфейс – логічна структура, з якої можна читати та записувати мережний трафік у вигляді пакетів [14]. В контексті програми існує два інтерфейси – зовнішній (зовнішня мережа) та внутрішній (внутрішня мережа). Аналізатор під'єднується до цих двох інтерфейсів та передає трафік між ними в обох напрямках. Без аналізатора з'єднання між інтерфейсами не існує, що є дуже важливим фактором, адже тільки в такому разі можливо здійснювати блокування аномального трафіку. Тобто, єдиним компонентом який передає трафік між зовнішнім та внутрішнім інтерфейсами є аналізатор. У разі виявлення DDoS атаки виводиться відповідне повідомлення, а UDP флуд – блокується (в залежності від конфігурації).

Схему роботи розробленого ПЗ зображено на рис. 3.1 (місце у схемі позначено елементом «Аналізатор»).

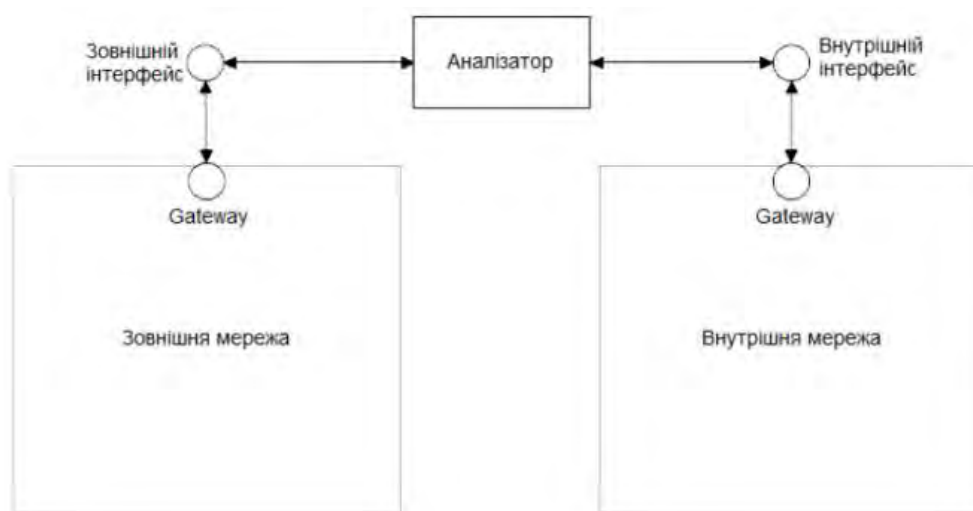


Рисунок 3.1 – Схема роботи системи

Параметри зовнішнього та внутрішнього інтерфейсів, а також параметри аналізатору, встановлює користувач за допомогою файлу конфігурації у форматі DOTENV. Яке правило – це `conf.env`, проте ім'я може бути різним, адже користувач обов'язково має передати файл конфігурації в якості аргументу запуску.

Програма представляє собою системне програмне забезпечення під Unix-подібні операційні системи (перш за все ОС на основі ядра Linux), тому не має графічного інтерфейсу користувача, який є непотрібним виходячи з призначення системи.

Назва програми – `watchdog` (від англ. DDoS WatchDog), маючи зміст «патрульного мережі від DDoS атак».

Параметри запуску аналізатора:

- `-c <config_file>` – передача файлу конфігурації (обов'язковий);
- `--help` – вивід довідки.

Узагальнено стрічка запуску виглядає наступним чином:

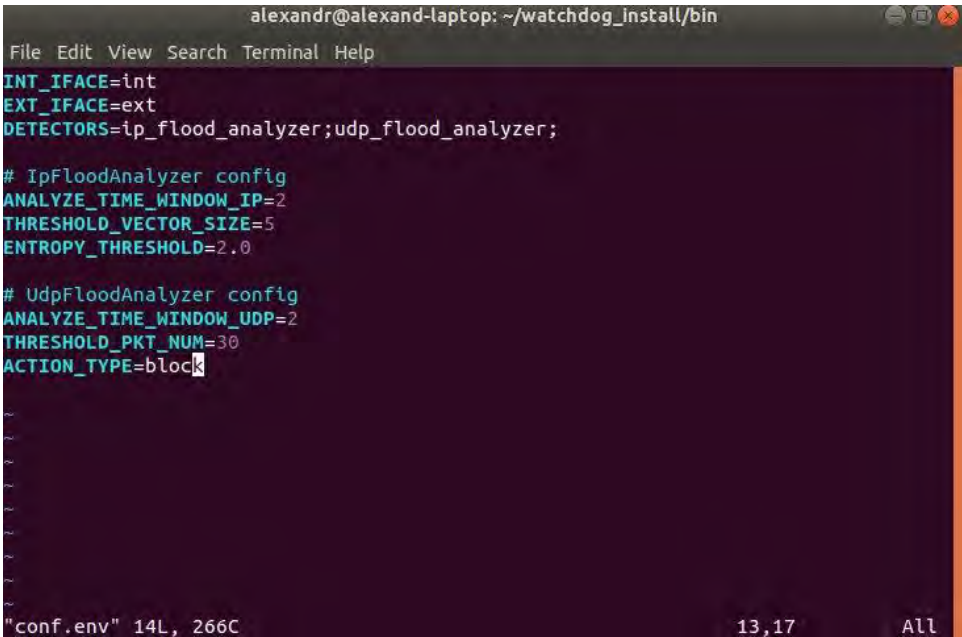
```
watchdog -c <config_file> [--help]
```

Формат файлу конфігурації – DOTENV, що представляє собою набір пар ключ-значення, де ключами є параметри системи. Є підтримка однорядкових коментарів (спеціальний символ «`#`» – початок коментаря). Список параметрів файлу конфігурації:

- `INT_IFACE` – ім'я внутрішнього інтерфейсу (обов'язковий);
- `EXT_IFACE` – ім'я зовнішнього інтерфейсу (обов'язковий);
- `DETECTORS` – список активованих детекторів, розділених крапкою з комою;
- `MAX_PRINT_THREADS` – максимальна кількість потоків виводу інформації о мережних пакетах (за замовчуванням 1000);
- `ANALYZE_TIME_WINDOW_IP` – довжина часового інтервалу аналізу IP-пакетів у секундах (за замовчуванням 2.0 секунди);

- THRESHOLD_VECTOR_SIZE – довжина вектору критичних значень (за замовчуванням 15 елементів);
- ENTROPY_THRESHOLD – порогове значення інформаційної ентропії (за замовчуванням 2.0);
- ANALYZE_TIME_WINDOW_UDP – довжина часового інтервалу аналізу UDP-пакетів у секундах (за замовчуванням 2.0 секунди);
- THRESHOLD_PKT_NUM – порогове значення кількості послідовних UDP-пакетів з однаковими значеннями контрольних сум (за замовчуванням 30 пакетів);
- ACTION_TYPE – дія UDP-флуд детектора у разі виявлення аномалії (за замовчуванням «alert»).

Приклад файлу конфігурації наведено на рис. 3.2.



```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
INT_IFACE=int
EXT_IFACE=ext
DETECTORS=ip_flood_analyzer;udp_flood_analyzer;

# IpFloodAnalyzer config
ANALYZE_TIME_WINDOW_IP=2
THRESHOLD_VECTOR_SIZE=5
ENTROPY_THRESHOLD=2.0

# UdpFloodAnalyzer config
ANALYZE_TIME_WINDOW_UDP=2
THRESHOLD_PKT_NUM=30
ACTION_TYPE=block

"conf.env" 14L, 266C 13,17 All
```

Рисунок 3.2 – Файл конфігурації

Функціонал програми зводиться до аналізу мережного трафіку на присутність атак типу DDoS. Процес роботи можна розділити на два потоки виконання:

- передача мережного трафіку із зовнішньої мережі до внутрішньої;

- передача мережного трафіку із внутрішньої мережі до зовнішньої.

Діаграму потоку даних із зовнішнього інтерфейсу зображено на рис. 3.3 згідно з нотацією UML [15].

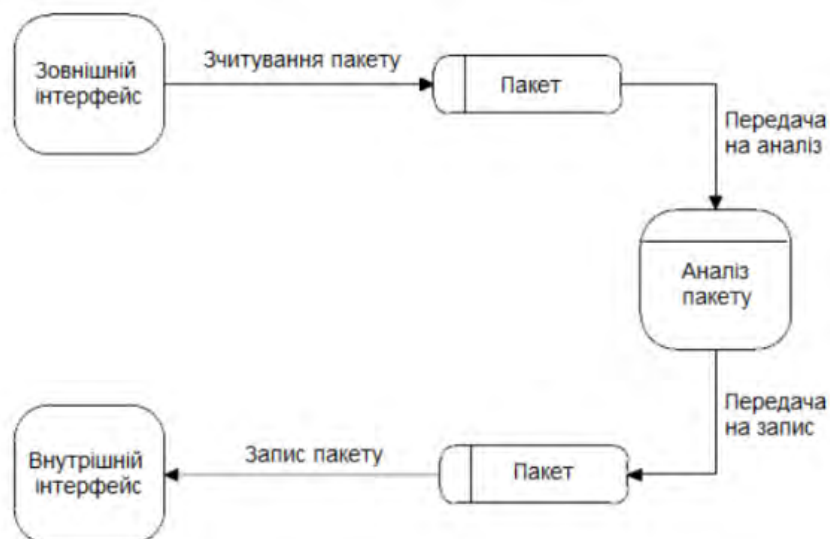


Рисунок 3.3 – Діаграма потоку даних із зовнішнього інтерфейсу до внутрішнього

Діаграму потоку даних із внутрішнього інтерфейсу зображено на рис. 3.4 згідно з нотацією UML [15].



Рисунок 3.4 – Діаграма потоку даних із внутрішнього інтерфейсу до зовнішнього

Концептуальна різниця між потоками тільки в тому, що перший має на меті здійснення аналізу та блокування мережного трафіку, а другий – лише передачу

трафіку без здійснення жодних дій. Це обумовлено тим, що не має сенсу аналізувати трафік з внутрішньої мережі, адже він є довіреним за своєю суттю. Проте, в архітектуру аналізатора закладено можливість розширення з додаванням процесу аналізу й для трафіку з внутрішньої мережі також.

Система складається з двох модулів – ядро та детектори. Ядро містить у собі основні, базові, компоненти системи, що формують каркас функціональності для компонентів з модуля детекторів. Список компонентів модуля ядра системи:

- компонент «watchdog» – основний компонент ініціалізації інших компонентів системи та керування ними;
- компонент «thread_control» – компонент управління потоками передачі мережного трафіку між інтерфейсами;
- компонент «detector_manager» – компонент ініціалізації та управління детекторами через інтерфейс «detector»;
- компонент «inline_set» – компонент, що інкапсулює логіку ініціалізації та керування парами зв'язаних інтерфейсів;
- компонент «sniffer» – компонент, що інкапсулює логіку ініціалізації та керування мережним інтерфейсом;
- компонент «logger» – компонент, відповідальний за вивід всіх повідомлень;
- компонент «config» – компонент, що інкапсулює логіку зчитування, збереження та доступу до конфігурації системи.

Список компонентів модуля детекторів (власне – список детекторів):

- компонент «network_analyzer» – компонент, що інкапсулює логіку виводу інформації про структуру мережних пакетів;
- компонент «ip_flood_analyzer» – компонент, що інкапсулює логіку виявлення аномалій мережного трафіку на основі розрахунку інформаційної ентропії;

– компонент «udp_flood_analyzer» – компонент, що інкапсулює логіку виявлення UDP флуду на основі аналізу послідовностей UDP-пакетів з однаковими значеннями контрольних сум.

Модуль детекторів пов'язаний з модулем ядра через компонент «detector_manager», за допомогою інтерфейсу «detector».

Діаграму компонентів системи зображено на рис. 3.5 згідно з нотацією UML [16].

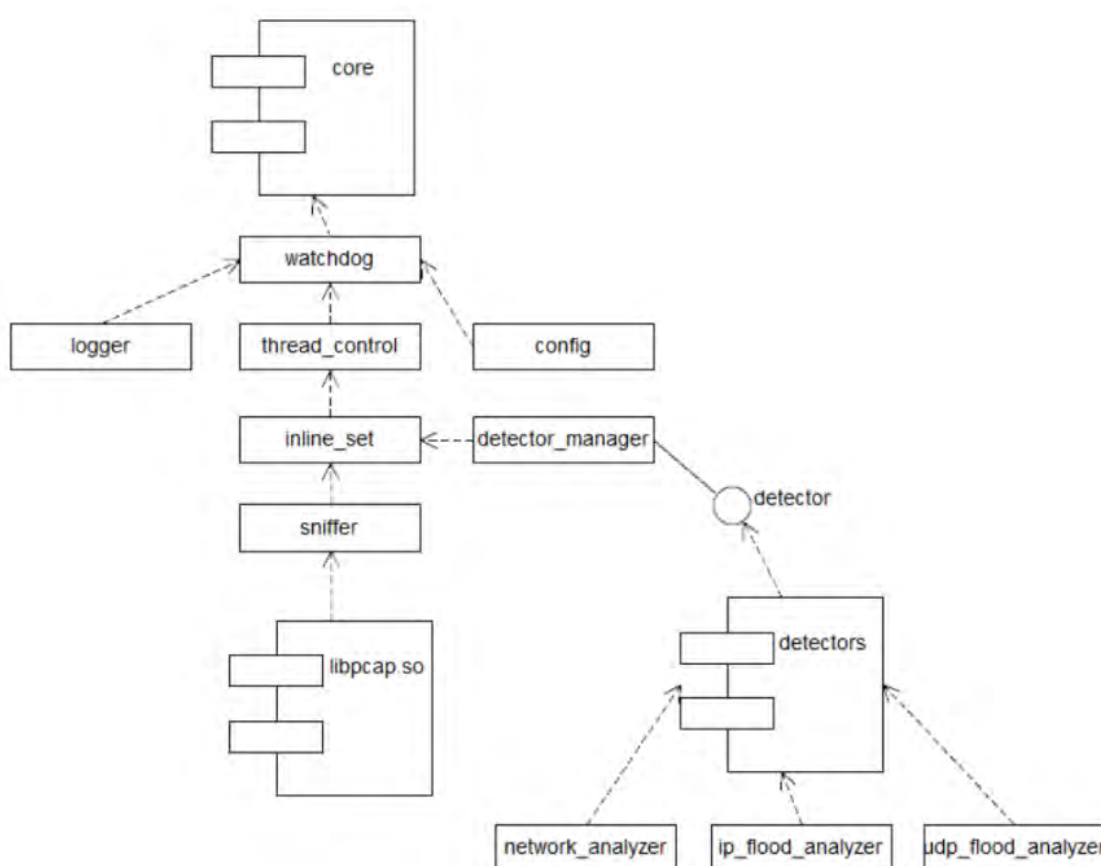


Рисунок 3.5 – Діаграма компонентів системи

Після компіляції, програма представляє собою цілісний виконавчий файл. Проте в архітектурі компонентів та зв'язків між ними, закладено можливість до подальшого розширення та абстракції функціоналу, що дасть змогу реалізувати детектори як окремі програмні компоненти (динамічні або статичні бібліотеки), незалежні від модуля ядра, через загальний програмний інтерфейс «detector».

3.2 Аналіз програмних засобів

3.2.1 Обґрунтування використання мови програмування C++

C++ – це мова програмування високого рівня, яка підтримує декілька основних парадигм програмування: процедурну, узагальнену та об'єктно-орієнтовану.

При створенні C++ ставили на меті збереження сумісності з мовою програмування C. Програми, розроблені на C, будуть працювати і з C++ компілятором. C++ та C мають однаковий базовий синтаксис.

Нововведеннями C++ у порівнянні з C є:

- об'єктно-орієнтоване програмування та класи;
- узагальнене програмування, шаблони;
- розширення компонентів стандартної бібліотеки;
- нові типи даних;
- обробка виняткових ситуацій;
- використання просторів імен;
- можливість перевантаження операторів;
- можливість перевантаження імен функцій;
- додано нові вбудовані функції;
- додано оператори управління пам'яттю.

Мова C++ є компільованою, тобто код має пройти процес компіляції, зв'язування та перетворення на бінарний виконавчий файл. Компілятори залежать від Асемблеру конкретної операційної системи, і представляють собою пряме перетворення коду на C++ у код інструкцій на мові Асемблеру конкретної архітектури. Саме тому програмне забезпечення, написане на C++ є дуже універсальним та оптимізованим. Найменші розміри виконавчих файлів, що досягається шляхом низького рівня базових компонентів C++, відсутність необхідності супроводжувати програму разом з транслятором завдяки процесу компіляції, адже виконавчий файл вже представляє собою повний набір інструкцій

і даних для конкретної платформи. Гнучкий та потужний менеджмент пам'яті дозволяє максимально ефективно використовувати оперативну пам'ять. Швидкість виконання інструкцій, знову ж таки, обумовлена компіляцією та низьким рівнем базових інструкцій мови [17].

C++ є основною мовою програмування в домені мережного програмування через високу ефективність, адже швидкодія та економія пам'яті є одним з головних параметрів розроблюваної системи. Так як передача трафіку між зовнішньою та внутрішньою мережами здійснюється через аналізатор, що само по собі створює додаткову затримку в передачі, будь-які додаткові витрати є недопустимими.

Також важливим фактором вибору C++ як мови програмування є наявність потужної та ефективної бібліотеки для роботи з мережею `libpcap`.

Використовується стандарт мови C++11, компілятор – GNU/g++.

3.2.2 Обґрунтування використання бібліотеки `libpcap`

`libpcap` (англ. Packet Capture Library) – це бібліотека, написана на мові програмування C, яка інкапсулює логіку роботи з мережним стеком операційної системи та дозволяє створювати програми для аналізу мережного трафіку. Вона може використовуватися як для зчитування пакетів з мережі так і для їх запису (відправлення) [18].

Так як бібліотека дозволяє використовувати зручний інтерфейс роботи з мережею і є високоефективною, її було обрано як компонент базової роботи з мережним обладнанням та мережним стеком операційної системи.

3.2.3 Обґрунтування використання генератора сценаріїв складання `Cmake`

`Cmake` – це крос-платформовий генератор сценаріїв складання з відкритим вихідним кодом. Він не займається безпосередньо складанням готового рішення, а

лише генерує файлу управління складанням (сценарії) з файлів CMakeLists.txt. Для Unix-подібних операційних систем це Makefiles [19].

Перевагами Cmake є:

- сценаріїв є простою;
- є засоби розширення функціональності;
- мінімум залежностей;
- кешування;
- крос-компіляція;
- підтримка створення файлів складання для великої кількості систем і

компіляторів.

Так як складання готового рішення «вручну» або за допомогою Makefiles напряду є незручним та складним процесом, з можливістю допущення помилок через значну кількість вихідних файлів проекту, використання системи генерації сценаріїв складання є обґрунтованим. Саме Cmake було обрано в якості такої системи через згадані вище переваги.

Для зручності виклику команди Cmake та її налаштування, використовується власно створений shell-скрипт «configure_cmake.sh».

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Детальний опис компонентів системи

Як було описано у попередньому розділі, програма складається з двох модулів, кожен з яких має у своєму складі окремі функціональні компоненти. Кожен компонент має чітке призначення та представляє собою об'єкт певного класу. Тобто, було використано принцип інкапсуляції об'єктно-орієнтованого дизайну. Логіка кожного окремого класу, а також логіка зв'язків та відношень між ними, формує загальну логіку роботи всієї системи.

Розглянемо компоненти модуля ядра системи. Перший та головний компонент всієї системи це «watchdog», що інкапсулює саму програму як логічний об'єкт. Ініціалізація та потоки виконання всієї програми починаються зі створення об'єкту цього класу. Діаграму класу зображено на рис. 4.1.

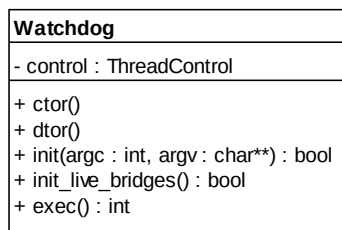


Рисунок 4.1 – Діаграма класу «Watchdog»

Компонент «thread_control» представлено класом ThreadControl, що інкапсулює логіку запуску та керування потоками передачі та аналізу трафіку між мережними інтерфейсами. Діаграму класу зображено на рис. 4.2.

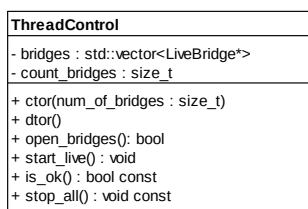


Рисунок 4.2 – Діаграма класу «ThreadControl»

Компонент «detector_manager» представлено класом «DetectorManager», що інкапсулює логіку ініціалізації ланцюгу обов’язків з детекторів, їх конфігурації та виконання. Увесь трафік, що передається між мережними інтерфейсами, проходить через цей компонент, який послідовно виконує логіку кожного детектора на кожному мережному пакеті через інтерфейс «Detector». Об’єкт класу «DetectorManager» не створюється, клас представляє собою інтерфейс із статичних методів до внутрішньої реалізації компоненту. Це зроблено з метою оптимізації та використання thread_local сховища послідовностей ініціалізованих детекторів, що дає можливість не піклуватися про синхронізацію доступу до цих сховищ у багатопотоковому середовищі, що також є оптимізацією часу виконання. Діаграму класу зображено на рис. 4.3.

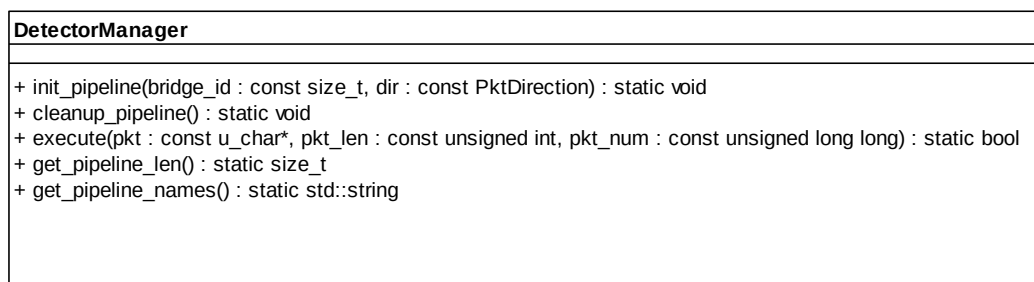


Рисунок 4.3 – Діаграма класу «DetectorManager»

Компонент «inline_set» інкапсулює логіку ініціалізації та керування парою зв’язаних мережних інтерфейсів. Він складається з двох класів – «InlineSet», що представляє, власне, пару зв’язаних інтерфейсів як логічний елемент системи, та «LiveBridge», що інкапсулює мережний міст між двома фізичними інтерфейсами як логічний елемент системи, використовуючи пару пов’язаних об’єктів-нащадків класу «InlineSet» (один для напрямку «від зовнішнього до внутрішнього» а другий – «від внутрішнього до зовнішнього»). Саме об’єкт класу «LiveBridge» запускає роботу пакетних потоків на відповідній парі мережних інтерфейсів в обох напрямках. Також, клас здійснює підрахунок кількості усіх мережних пакетів що пройшли через пару пов’язану інтерфейсів. Діаграму класів зображено на рис. 4.4.

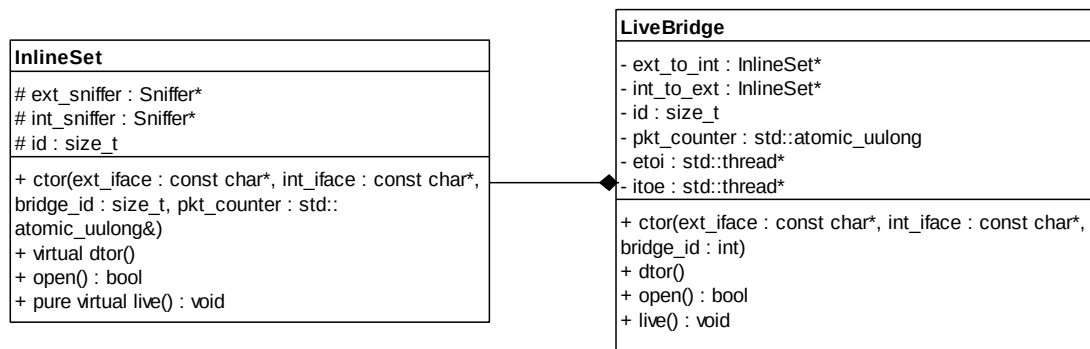


Рисунок 4.4 – Діаграма класів компоненту «inline_set»

Компонент «sniffer» представлено класом «Sniffer», що представляє собою окремий мережний інтерфейс як логічний елемент системи, з визначеними операціями зчитування та запису. Клас використовує інтерфейс бібліотеки libpcap для роботи з мережним стеком операційної системи та, з точки зору дизайну, представляє собою об'єкт-фасад до цієї бібліотеки, тобто – зручний інтерфейс у контексті розроблюваної системи для взаємодії з мережним інтерфейсом без необхідності знати деталі роботи з інтерфейсом libpcap. Діаграму класу зображено на рис. 4.5.

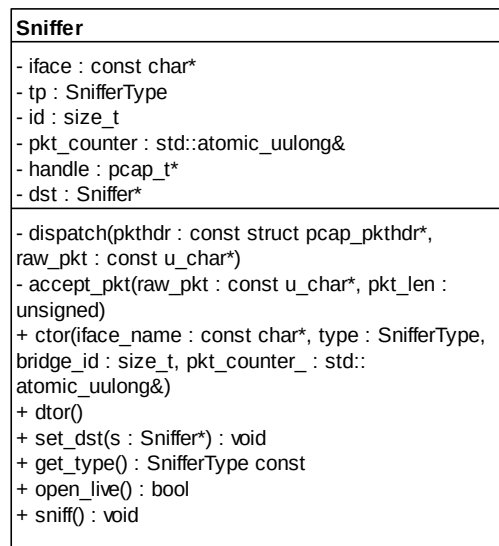


Рисунок 4.5 – Діаграма класу «Sniffer»

Компонент «logger» представлено класом «Logger», що інкапсулює логіку виводу повідомлень до стандартного потоку виводу stdout. Об'єкт цього класу не

створюється, клас представляє собою лише інтерфейс статичних методів до внутрішньої логіки виводу повідомлень. Діаграму класу зображено на рис. 4.6.

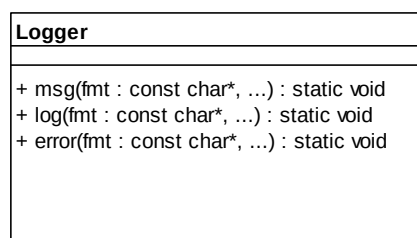


Рисунок 4.6 – Діаграма класу «Logger»

Компонент «config» представлено класом «Config», що інкапсулює логіку завантаження, збереження та доступу до конфігурації системи. Так як конфігурація є загальною для всіх компонентів системи й об'єкт класу має бути ініціалізований лише один раз, було використано шаблон проектування «Одинак» (англ. «Singleton»). Діаграму класу зображено на рис. 4.7.

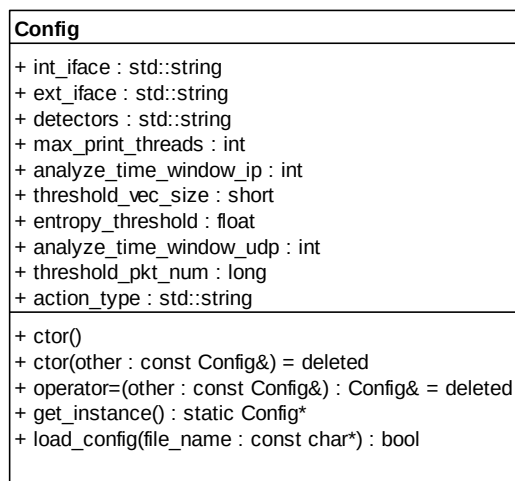


Рисунок 4.7 – Діаграма класу «Config»

Розглянемо структуру компонентів модуля детекторів (власне – структуру детекторів). Головне місце тут займає інтерфейс «Detector», через який компонент «detector_manager» ініціалізує та керує детекторами. Кожен детектор мусить наслідувати цей інтерфейс. Діаграму класу зображено на рис. 4.8.

Detector
<pre># name : const char* # bridge_id : const size_t # dir : const PktDirection</pre>
<pre>+ ctor(name : const char*, bridge_id : const size_t, dir : const PktDirection) + virtual dtor() + virtual analyze(pkt : const u_char*, pkt_len : const unsigned int, pkt_num : const unsigned long long) : bool + get_name() : const char* const</pre>

Рисунок 4.8 – Діаграма класу-інтерфейсу «Detector»

Перший детектор – це «network_analyzer», представлений класом «NetworkAnalyzer», що інкапсулює логіку виводу інформації о перехоплених мережних пакетах. Цей вивід можна використовувати у цілях моніторингу мережної активності а також для налагоджування програми (англ. debugging). Діаграму класу зображено на рис. 4.9.

Наступний детектор – це «ip_flood_analyzer», представлений класом «IpFloodAnalyzer», що інкапсулює логіку аналізу мережного трафіку на присутність атак типу DDoS методом розрахунку інформаційної ентропії на сталих інтервалах часу. Діаграму класу зображено на рис. 4.10.

Останній реалізований детектор це «udp_flood_analyzer», представлений класом «UdpFloodAnalyzer», що інкапсулює логіку аналізу мережного трафіку на присутність UDP флуду як виду DDoS атаки методом аналізу послідовностей UDP-пакетів з однаковими значеннями контрольних сум. Діаграму класу зображено на рис. 4.11.

NetworkAnalyzer
<pre>- max_threads : const int - num_threads : std::atomic_ulong</pre>
<pre>+ ctor(bridge_id : const size_t, dir : const PktDirection, max_threads : const int) + dtor() + analyze(pkt : const u_char*, pkt_len : const unsigned int, pkt_num : const unsigned long long) : bool + override - print(pkt : const u_char*, pkt_num : const unsigned long long)</pre>

Рисунок 4.9 – Діаграма класу «NetworkAnalyzer»

IpFloodAnalyzer
- ip_dict : IpDict (std::unordered_map<uint32_t, long long>) - threshold_vec : std::vector<short> - analyze_time_window : const int - threshold_vec_size : const short - vector_pos : short - pkt_count : long long - is_traffic : bool - entropy_threshold : const float - mtx : std::mutex - evaluation_thrd : std::thread
+ ctor(bridge_id : const size_t, dir : const PktDirection, time_window : int, vector_size : short, threshold : float) + dtor() + analyze(pkt : const u_char*, pkt_len : const unsigned int, pkt_num : const unsigned long long) : bool override - insert_ip(ip : uint32_t) : void - clear_dict() : void - evaluate_entropy() : void - evaluate() : void - detect_anomaly() : void

Рисунок 4.10 – Діаграма класу «IpFloodAnalyzer»

UdpFloodAnalyzer
- udp_flow_dict : UdpFlowDict (std::unordered_map<uint32_t, std::unordered_map<uint16_t, unsigned long>>) - checksum_dict : UdpCheckSumDict (std::unordered_map<uint32_t, std::unordered_map<uint16_t, uint16_t>>) - blocked_udp : UdpBlockedMap (std::unordered_multimap<uint32_t, uint16_t>) - threshold_pkt_num : const unsigned long - analyze_time_window : const int - action_type : ActionType - mtx_flow : std::mutex - mtx_blocked_list : std::mutex - evaluation_thrd : std::thread
+ ctor(bridge_id : const size_t, dir : const PktDirection, time_window : const int, threshold : const long, action_type : const std::string&) + dtor() + analyze(pkt : const u_char*, pkt_len : const unsigned int, pkt_num : const unsigned long long) : bool override - is_blocked(ip : const uint32_t, port : const uint16_t) : bool - insert_udp(ip : const uint32_t, port : const uint16_t, checksum : const uint16_t) : void - clear_dict() : void - evaluate() : void - detect_anomaly() : void - action(ip : const uint32_t, port : const uint16_t) : void - alert(ip : const uint32_t, port : const uint16_t) : void - block(ip : const uint32_t, port : const uint16_t) : void

Рисунок 4.11 – Діаграма класу «UdpFloodAnalyzer»

Діаграму класів модуля ядра зображено на рис. 4.12 згідно з нотацією UML [20].

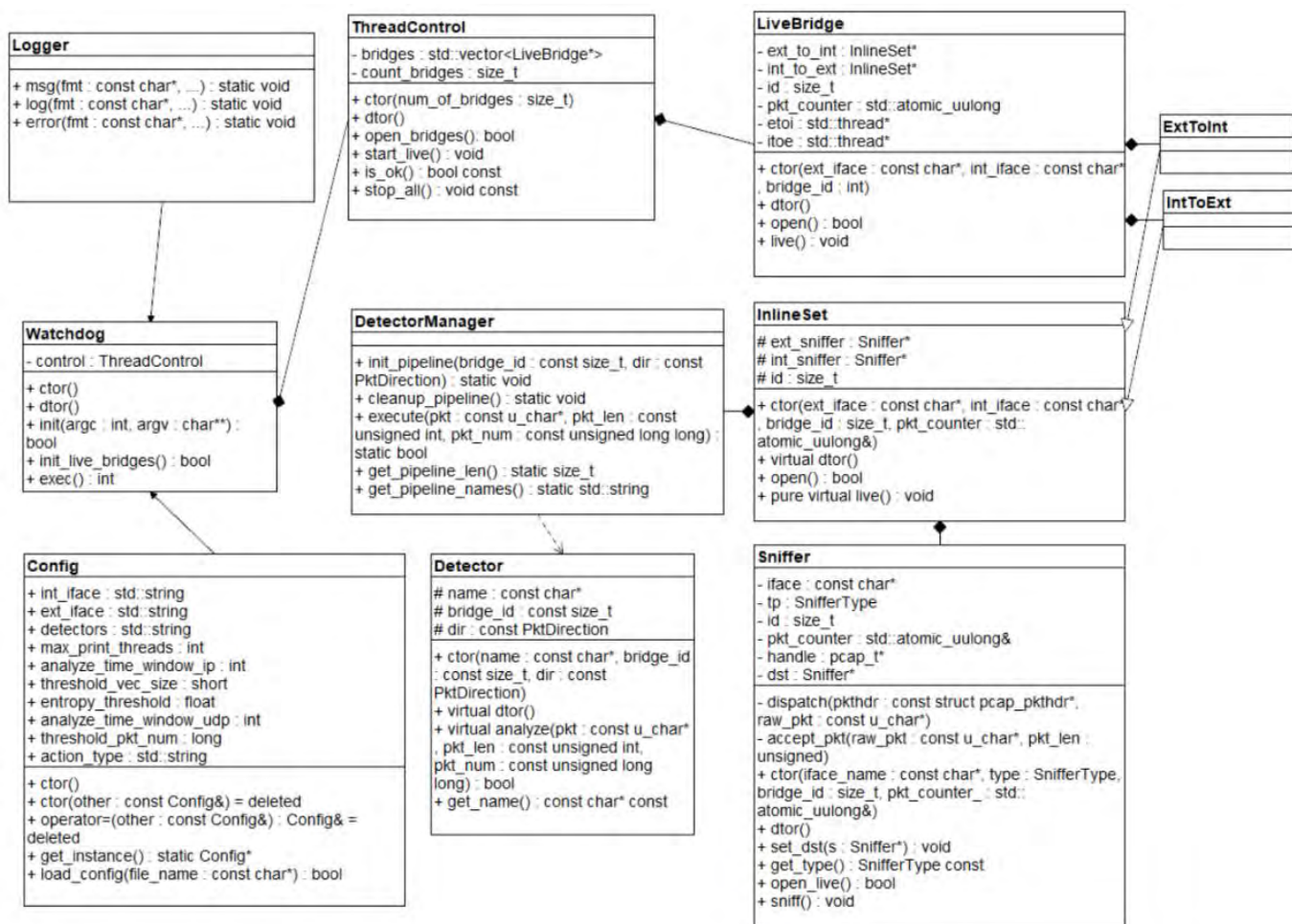


Рисунок 4.12 – Діаграма класів модуля ядра системи

Діаграму класів модуля детекторів зображено на рис. 4.13 згідно з нотацією UML [20].

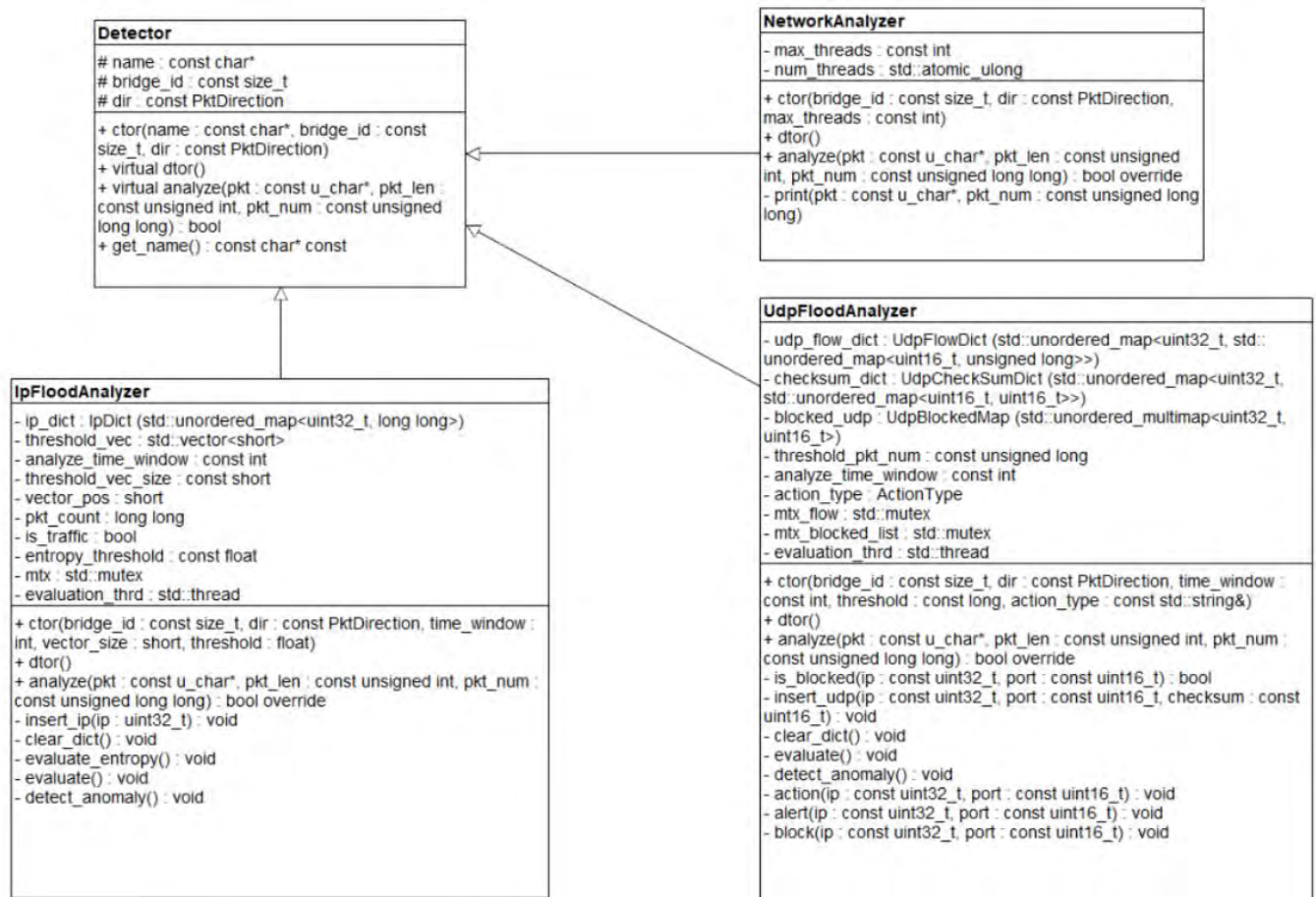


Рисунок 4.13 – Діаграма класів модуля детекторів системи

4.2 Опис потокової архітектури системи

Розроблена система накладає мінімальну мережну затримку під час аналізу трафіку завдяки використанню багатопоточної архітектури [21]. Кожний потік мережного трафіку між інтерфейсами в обох напрямках (із зовнішнього інтерфейсу до внутрішнього й у зворотному) оброблюється в окремому потоці виконання задля підвищення ефективності системи. Також це зроблено з точки зору дизайну системи, адже існує можливість накладання логіки аналізу на кожний окремий потік. Такі потоки керуються компонентом «thread_control».

Також, кожен з детекторів створює власний додатковий потік для аналізу трафіку. Це зроблено з точки зору оптимізації та особливостей логіки алгоритмів виявлення аномалій. Такі потоки керуються кожним окремим детектором.

Синхронізація пакетних потоків досягається використанням «слотів потоків» (англ. thread slots), а потоків аналізу – за допомогою м'ютексів (англ. mutex) [21]. Слоти – це вектор, кожний елемент якого представляє собою певний контекст потоку та точку керування ним. Кожен потік має власний ідентифікатор, що відповідає індексу у векторі. Через єдиний програмний інтерфейс здійснюється доступ до слотів (вектору) за ідентифікатором потоку, без необхідності використання додаткової синхронізації. Потоки аналізу навпаки, використовують синхронізацію на основі м'ютексів. Це обумовлено необхідністю використання розподілених даних між пакетними потоками та потоками аналізу. Варто зазначити, що пакетний потік не блокується потоком аналізу, а лише копіює необхідний контекст отриманого пакету. Далі аналіз відбувається вже в окремому потоці.

Схему потокової архітектури системи зображено на рис. 4.14 у вигляді діаграми послідовностей згідно з нотацією UML [22].

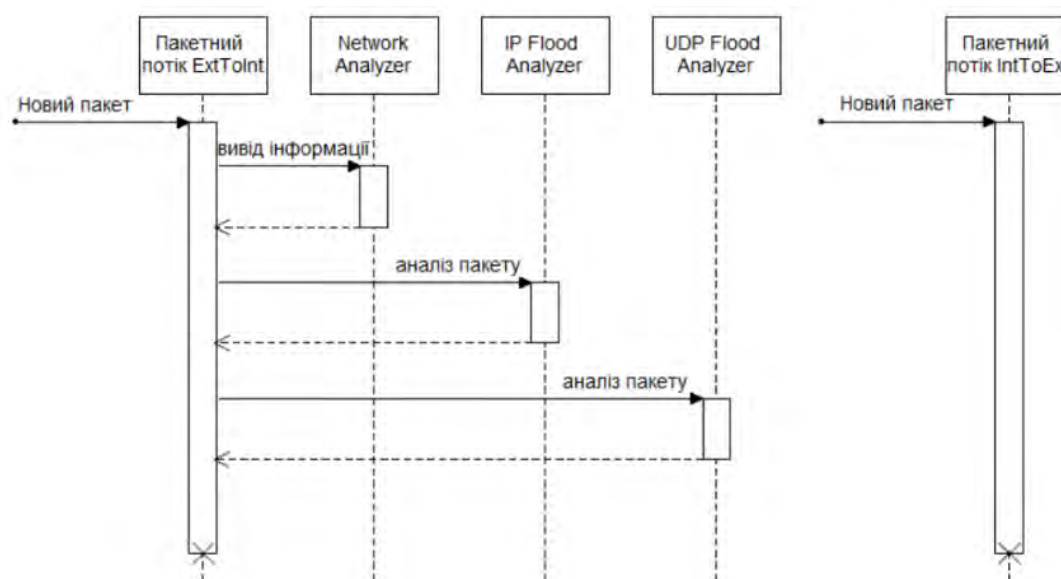


Рисунок 4.14 – Схема потокової архітектури системи

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Керівництво системного програміста

5.1.1 Призначення програми та системні вимоги

Призначенням програмного продукту є аналіз мережного трафіку на присутність атак типу DDoS. У разі виявлення атаки виводиться відповідне повідомлення а також, за можливості, аномальний трафік блокується. Розроблене ПЗ можна використовувати для захисту внутрішньої мережі суб'єкта використання від загрози DDoS атак.

Характеристики комп'ютера, необхідні для стабільної та швидкої роботи програми, повинні задовольняти вимогам, зазначеним в табл. 5.1.

Таблиця 5.1 – Системні вимоги

Складова комп'ютера	Характеристика
CPU	Intel або AMD 2-core, 2.6GHz або краще
RAM	2Gb або більше
Простір на диску	50Mb
Додаткове ПЗ	libpcap
Операційна система	Unix-based (Linux, MacOS)

5.1.2 Характеристики програми

Програма не має графічного інтерфейсу користувача, тільки інтерфейс командного рядка.

Для запуску необхідно передати шлях до файлу конфігурації. Обов'язково запускати від імені адміністратора, так як для доступу до мережних інтерфейсів необхідні відповідні привілеї.

Конфігурація програми здійснюється шляхом внесення змін до файлу конфігурації `conf.env` у форматі `DOTENV`. Його зміст викладено у табл. 5.2.

Таблиця 5.2 – Зміст файлу конфігурації `conf.env`

Поле	Значення
<code>INT_IFACE</code>	Ім'я внутрішнього інтерфейсу (обов'язковий)
<code>EXT_IFACE</code>	Ім'я зовнішнього інтерфейсу (обов'язковий)
<code>DETECTORS</code>	Список активованих детекторів, розділених крапкою з комою
<code>MAX_PRINT_THREADS</code>	Максимальна кількість потоків виводу інформації о мережних пакетах (за замовчуванням 1000)
<code>ANALYZE_TIME_WINDOW_IP</code>	Довжина часового інтервалу аналізу IP- пакетів у секундах (за замовчуванням 2.0 секунди)
<code>THRESHOLD_VECTOR_SIZE</code>	Довжина вектору критичних значень (за замовчуванням 15 елементів)
<code>ENTROPY_THRESHOLD</code>	Порогове значення інформаційної ентропії (за замовчуванням 2.0)
<code>ANALYZE_TIME_WINDOW_UDP</code>	Довжина часового інтервалу аналізу UDP- пакетів у секундах (за замовчуванням 2.0 секунди)
<code>THRESHOLD_PKT_NUM</code>	Порогове значення кількості послідовних UDP-пакетів з однаковими значеннями контрольних сум (за замовчуванням 30 пакетів)

Продовження таблиці 5.2

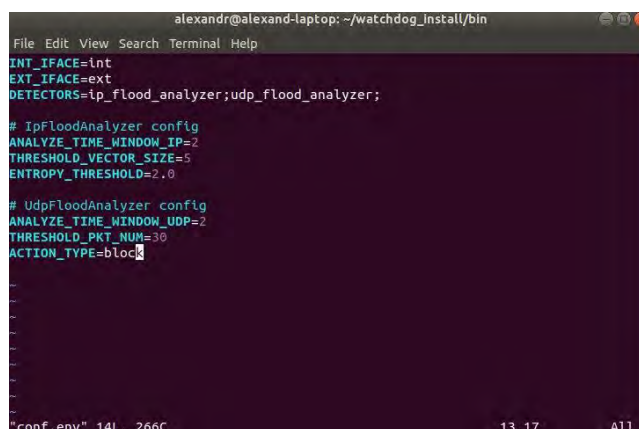
Поле	Значення
ACTION_TYPE	Дія UDP-флуд детектора у разі виявлення аномалії (за замовчуванням «alert»)
#	Символ початку однорядкового коментаря

Команда запуску програми має наступний вигляд:

```
watchdog -c <config_file> [--help]
```

Де <config_name> – повне ім'я файлу конфігурації.

Приклад файлу конфігурації зображено на рис. 5.1.



```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
INT_IFACE=int
EXT_IFACE=ext
DETECTORS=ip_flood_analyzer;udp_flood_analyzer;

# IpFloodAnalyzer config
ANALYZE_TIME_WINDOW_IP=2
THRESHOLD_VECTOR_SIZE=5
ENTROPY_THRESHOLD=2.0

# UdpFloodAnalyzer config
ANALYZE_TIME_WINDOW_UDP=2
THRESHOLD_PKT_NUM=30
ACTION_TYPE=block

"conf.env" 14L, 266C 13,17 All
```

Рисунок 5.1 – Файл конфігурації conf.env

Перелік використаних бібліотек (список залежностей):

- libpcap – бібліотека для роботи з мережним стеком операційної системи;
- libstdc – стандартна бібліотека мови програмування C;
- libstdc++ – стандартна бібліотека мови програмування C++;
- libpthread – бібліотека роботи з потоками виконання.

5.1.3 Підготовка програми до запуску

Щоб підготувати систему до роботи необхідно виконати наступні кроки:

- обрати комп'ютер, що відповідає системним вимогам (табл. 5.1) та розмістити програму на ньому;
- переконатися у доступності пари мережних інтерфейсів, які будуть поєднуватися (інтерфейси не повинні бути з'єднані завчасно);
- переконатися у доступності привілеїв адміністратора;
- переконатися у доступності всіх необхідних бібліотек зі списку залежностей;
- налаштувати систему шляхом визначення файлу конфігурації `conf.env`, описаний у таблиці 5.2.

5.1.4 Вхідні і вихідні дані

У якості вхідних даних виступають пакети, зчитані з мережного інтерфейсу, та завантажений файл конфігурації для налаштування системи.

У якості вихідних даних виступають пакети, записані до мережного інтерфейсу, інформація щодо цих пакетів, а також повідомлення різних типів (повідомлення, лог, помилка), що виводяться до потоку стандартного виводу `stdout`.

Підтримувані мережні протоколи: Ethernet, ARP, Ipv4, ICMP, TCP, UDP.

5.2 Керівництво оператора

5.2.1 Призначення програми

Призначенням програмного продукту є аналіз мережного трафіку на присутність атак типу DDoS. У разі виявлення атаки виводиться відповідне повідомлення а також, за можливості, аномальний трафік блокується. Головна мета

програми – захист та попередження внутрішньої мережі суб'єкта використання від атак типу DDoS.

5.2.2 Умови виконання програми

Характеристики комп'ютера, необхідні для стабільної та швидкої роботи програми, мають задовольняти вимоги, зазначені в табл. 5.3.

Таблиця 5.3 – Системні вимоги

Складова комп'ютера	Характеристика
CPU	Intel або AMD 2-core, 2.6GHz або краще
RAM	2Gb або більше
Простір на диску	50Mb
Додаткове ПЗ	libpcap
Операційна система	Unix-based (Linux, MacOS)

5.2.3 Встановлення програми

Для розміщення програми на фізичному носію необхідно розпакувати архів з аналізатором і налаштувати конфігурацію. Після цього програма готова до використання.

5.2.4 Видалення програми

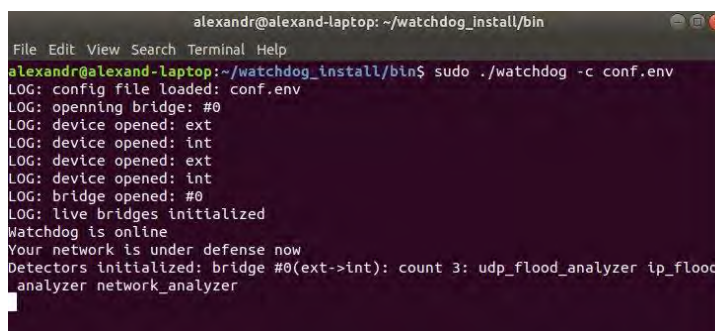
Для видалення програми з фізичного носія, необхідно видалити розпаковані файли програми з попереднього пункту.

5.2.5 Виконання програми

Для запуску програми, необхідно завчасно підготувати файл конфігурації та викликати команду для запуску з терміналу (рис. 5.2). Програма вже працює та не потребує жодних інших дій від користувача.

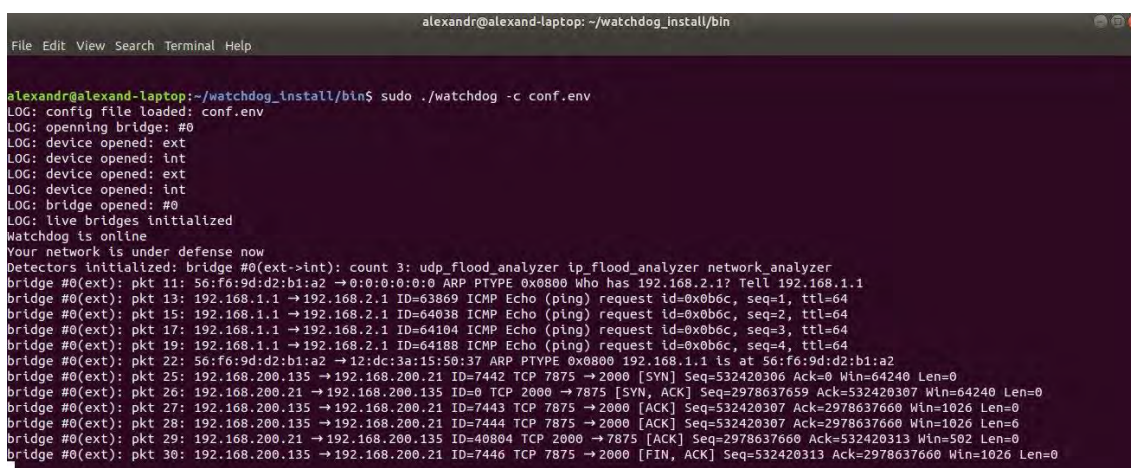
Якщо в конфігурації був вказаний детектор «network_analyzer», у потік стандартного виводу буде виводитися інформація про оброблювані пакети (рис. 5.3).

У разі виявлення DDoS атаки буде виведено відповідне повідомлення (рис. 5.4), після чого, системний адміністратор вручну чи автоматично мусить активувати заходи протидії. Наприклад – фільтрація по IP-адресам пакетів з бази даних IP-адрес легітимних користувачів, що формується під час нормальної роботи сервера.



```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 3: udp_flood_analyzer ip_flood_analyzer network_analyzer
```

Рисунок 5.2 – Запуск програми



```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 3: udp_flood_analyzer ip_flood_analyzer network_analyzer
bridge #0(ext): pkt 11: 56:f6:9d:d2:b1:a2 -> 0:0:0:0:0:0 ARP PTYPE 0x0000 Who has 192.168.2.1? Tell 192.168.1.1
bridge #0(ext): pkt 13: 192.168.1.1 -> 192.168.2.1 ID=63869 ICMP Echo (ping) request id=0x0b6c, seq=1, ttl=64
bridge #0(ext): pkt 15: 192.168.1.1 -> 192.168.2.1 ID=64038 ICMP Echo (ping) request id=0x0b6c, seq=2, ttl=64
bridge #0(ext): pkt 17: 192.168.1.1 -> 192.168.2.1 ID=64104 ICMP Echo (ping) request id=0x0b6c, seq=3, ttl=64
bridge #0(ext): pkt 19: 192.168.1.1 -> 192.168.2.1 ID=64188 ICMP Echo (ping) request id=0x0b6c, seq=4, ttl=64
bridge #0(ext): pkt 22: 56:f6:9d:d2:b1:a2 -> 12:dc:3a:15:50:37 ARP PTYPE 0x0000 192.168.1.1 is at 56:f6:9d:d2:b1:a2
bridge #0(ext): pkt 25: 192.168.200.135 -> 192.168.200.21 ID=7442 TCP 7875 -> 2000 [SYN] Seq=532420306 Ack=0 Win=64240 Len=0
bridge #0(ext): pkt 26: 192.168.200.21 -> 192.168.200.135 ID=0 TCP 2000 -> 7875 [SYN, ACK] Seq=2978637659 Ack=532420307 Win=64240 Len=0
bridge #0(ext): pkt 27: 192.168.200.135 -> 192.168.200.21 ID=7443 TCP 7875 -> 2000 [ACK] Seq=532420307 Ack=2978637660 Win=1026 Len=0
bridge #0(ext): pkt 28: 192.168.200.135 -> 192.168.200.21 ID=7444 TCP 7875 -> 2000 [ACK] Seq=532420307 Ack=2978637660 Win=1026 Len=0
bridge #0(ext): pkt 29: 192.168.200.21 -> 192.168.200.135 ID=40804 TCP 2000 -> 7875 [ACK] Seq=2978637660 Ack=532420313 Win=502 Len=0
bridge #0(ext): pkt 30: 192.168.200.135 -> 192.168.200.21 ID=7446 TCP 7875 -> 2000 [FIN, ACK] Seq=532420313 Ack=2978637660 Win=1026 Len=0
```

Рисунок 5.3 – Вивід пакетної інформації

```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
```

Рисунок 5.4 – Повідомлення про присутність DDoS атаки

У разі виявлення UDP-флуду, програма виведе відповідне повідомлення (рис. 5.5) або повідомлення про блокування (рис. 5.6), якщо це налаштовано в конфігурації системи.

```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 53426
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 59410
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 56283
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 49199
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 56798
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 55106
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 52724
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 46577
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 41851
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 37371
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 37371
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 49199
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 56798
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 41851
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 46577
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 59410
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 53426
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 55106
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 56283
bridge #0(ext): udp_flood_analyzer: ALERT: anomaly detected: IP 192.168.1.1, PORT 52724
```

Рисунок 5.5 – Повідомлення про присутність UDP-флуду

```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 52724
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 53426
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 59410
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 55106
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 56283
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 37371
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 46577
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 41851
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 56798
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 49199
```

Рисунок 5.6 – Повідомлення про блокування UDP-флуду

Для завершення програми достатньо зупинити процес стандартною комбінацією клавіш Ctrl+C з терміналу або утилітою «kill», вказавши ID процесу аналізатора. Для знаходження ID процесу можна скористуватися командою «ps -A | grep watchdog».

5.3 Експериментальні дослідження

Дослідження розробленої системи можна поділити на дві групи – дослідження функціоналу на правильність та дослідження стійкості системи при довготривалій роботи (стрес-дослідження).

Був розроблений тестовий стенд на основі віртуальних мереж. Операційна система Linux дозволяє створювати віртуальні мережі та інтерфейси, які використовують той самий мережний стек операційної системи що й фізичні інтерфейси та мережі, тому отримані результати на цьому стенді будуть актуальними й для справжнього, не віртуального, середовища. Схему тестового стенду зображено на рис. 5.7. Для його створення було розроблено shell-скрипт (init_vnet.sh), що використовує утиліту ip [23]. В якості віртуальних мереж використовується поняття мережного простору (команда «ip netns add»).

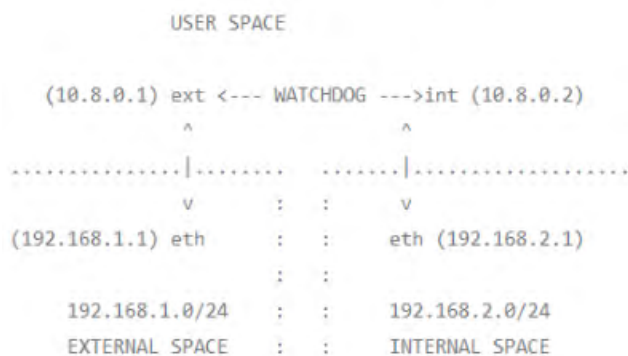


Рисунок 5.7 – Схема тестового стенду

Для перевірки правильності роботи системи, було підготовлено зразки легітимного трафіку для TCP, UDP та декількох інших протоколів, а також – зразок DDoS трафіку, згенерованого програмою hping3 та UDP-флуду, згенерованого програмою LOIC, опис яких згадувався у розділі 1.4 цієї роботи. Візуалізацію зразків трафіку зображено на рисунках нижче (рис. 5.8, 5.9, 5.10, 5.11, 5.12). Для генерації окремих зразків легітимного трафіку було використано програму для генерації мережного трафіку абсір.

The screenshot shows the tcpdump application interface. The main window displays a list of captured packets. The first packet is a SYN packet from 192.168.200.135 to 192.168.200.21. The details pane shows the packet structure: Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet details are as follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.200.135	192.168.200.21	TCP	66	7875 → 2000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000053	192.168.200.21	192.168.200.135	TCP	66	2000 → 7875 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.004573	192.168.200.135	192.168.200.21	TCP	60	7875 → 2000 [ACK] Seq=1 Ack=1 Win=262656 Len=0
4	0.004678	192.168.200.135	192.168.200.21	TCP	60	7875 → 2000 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=6
5	0.004690	192.168.200.21	192.168.200.135	TCP	54	2000 → 7875 [ACK] Seq=1 Ack=7 Win=64256 Len=0
6	2.738481	192.168.200.135	192.168.200.21	TCP	60	7875 → 2000 [FIN, ACK] Seq=7 Ack=1 Win=262656 Len=0
7	2.738675	192.168.200.21	192.168.200.135	TCP	54	2000 → 7875 [FIN, ACK] Seq=1 Ack=8 Win=64256 Len=0
8	2.742070	192.168.200.135	192.168.200.21	TCP	60	7875 → 2000 [ACK] Seq=8 Ack=2 Win=262656 Len=0
9	0.041825	192.168.200.135	192.168.200.21	TCP	66	7876 → 2000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
10	0.041865	192.168.200.21	192.168.200.135	TCP	66	2000 → 7876 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
11	0.047489	192.168.200.135	192.168.200.21	TCP	60	7876 → 2000 [ACK] Seq=1 Ack=1 Win=262656 Len=0
12	0.047526	192.168.200.135	192.168.200.21	TCP	1514	7876 → 2000 [ACK] Seq=1 Ack=1 Win=262656 Len=1460
13	0.047543	192.168.200.21	192.168.200.135	TCP	54	2000 → 7876 [ACK] Seq=1 Ack=1461 Win=64128 Len=0

The details pane also shows the packet structure: Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0. Ethernet II, Src: Dell_96:12:0e (ec:f4:bb:96:12:0e), Dst: Vmware_b4:90:14 (00:0c:29:b4:90:14). Internet Protocol Version 4, Src: 192.168.200.135, Dst: 192.168.200.21. Transmission Control Protocol, Src Port: 7875, Dst Port: 2000, Seq: 0, Len: 0.

Рисунок 5.8 – Легітимний TCP-трафік (tcp.pcap)

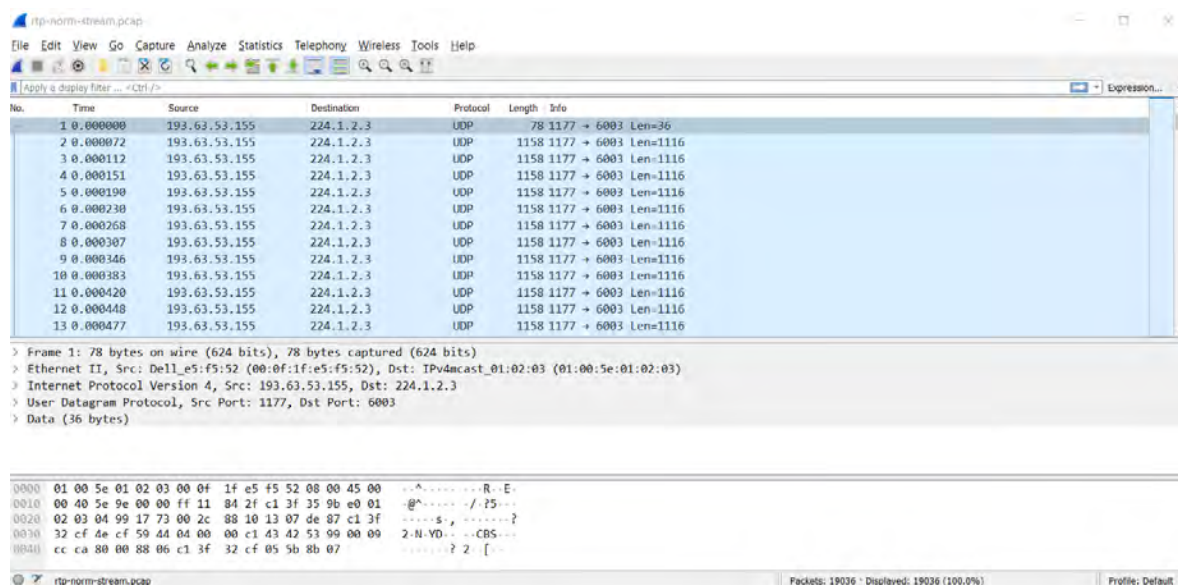


Рисунок 5.9 – Легітимний UDP-трафік (RTP-сесія, rtp-norm-stream.pcap)

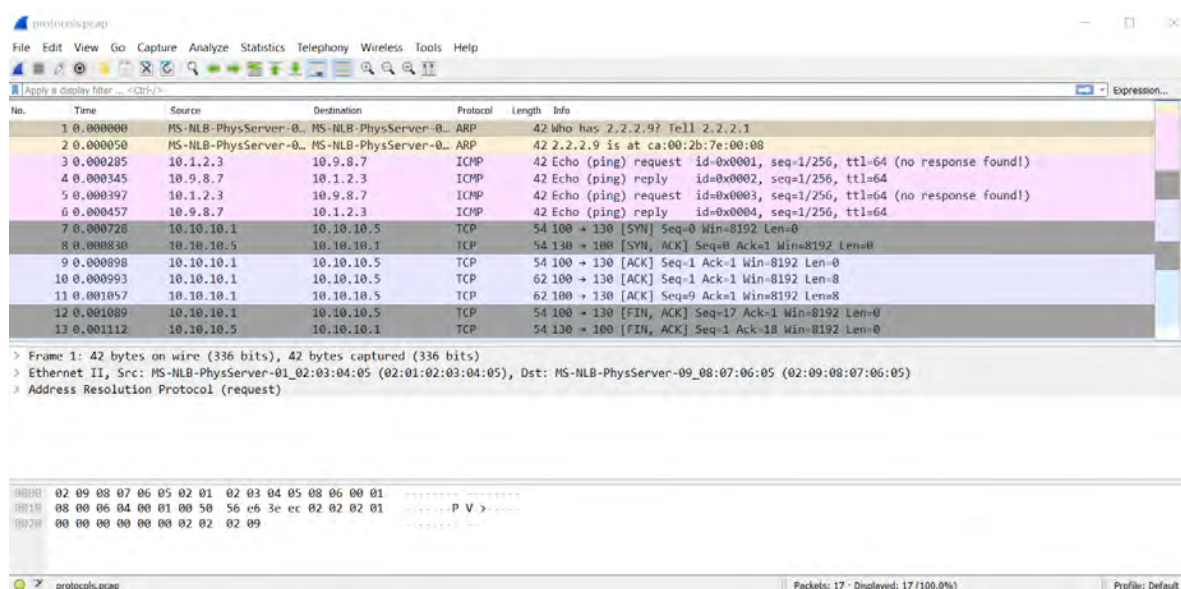


Рисунок 5.10 – Легітимний трафік декількох інших мережних протоколів, згенерований за допомогою програми abccir (protocols.pcap)

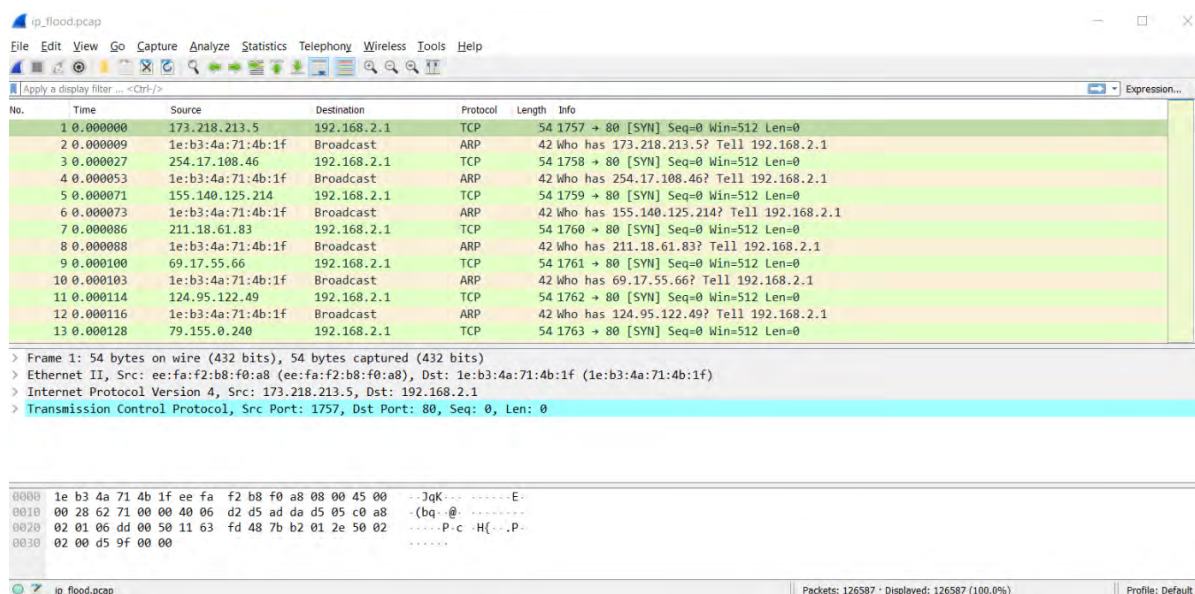


Рисунок 5.11 – DDoS трафік, згенерований програмою hping3 (ip_flood.pcap)

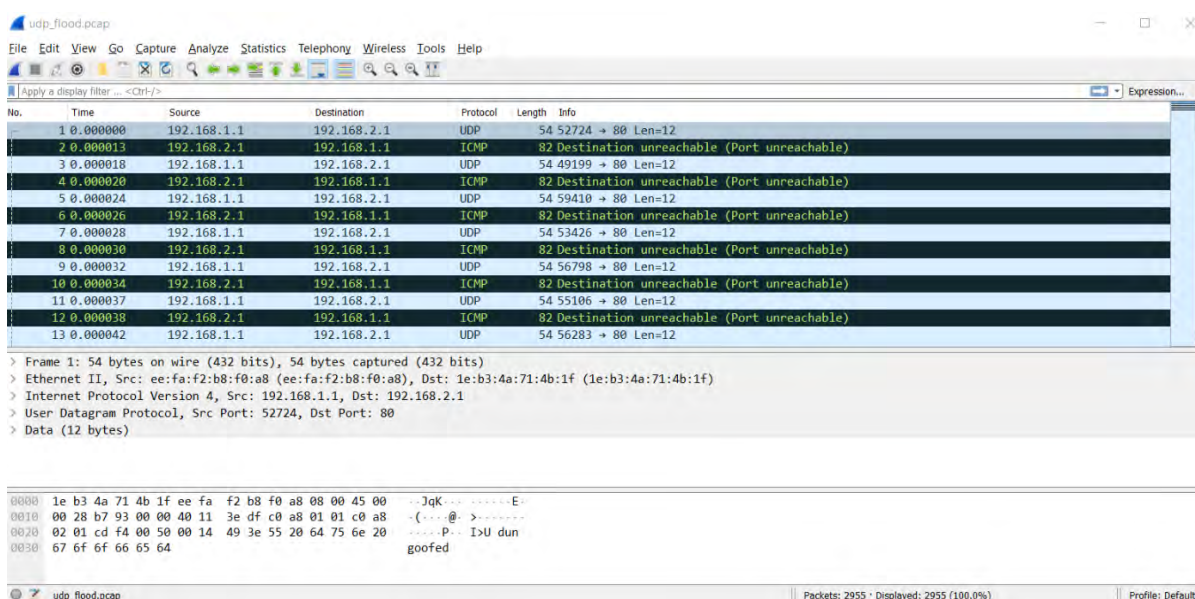


Рисунок 5.12 – UDP-флуд, згенерований програмою LOIC (udp_flood.pcap)

В результаті роботи програми з вказаними вище зразками мережного трафіку має спостерігатися наступна поведінка – система виявляє присутність DDoS атаки на зразках з аномальним трафіком, блокує його (якщо ця опція налаштована), та нічого не робить у разі проходження легітимного трафіку. Також тест має включати ситуацію змішаного легітимного та аномального трафіку. Запускати зразки варто з під віртуальних мереж, до яких під'єднано аналізатор наступною командою:

```
sudo ip netns exec <мережа> <команда>
```

Для відправлення зразків трафіку на відповідний мережний інтерфейс було розроблено декілька shell-скриптів, що використовують програму `tcpreplay` [24]:

```
sudo tcpreplay -i <інтерфейс> <файл зі зразком трафіку (.pcap)>
```

Дослідження системи на стійкість має на меті такий самий підхід, що й дослідження на правильність роботи системи, але з реальним прикладом використання, а також під час довготривалого використання з високою інтенсивністю DDoS атаки. Для цього було розроблено простий HTTP-сервер на Python, який буде ціллю атаки. У якості легітимного клієнта було використано HTTP-запити, згенеровані програмним забезпеченням `curl` в режимі реального часу. Для введення DDoS-навантаження було використано відповідне програмне забезпечення `hping3` та `LOIC` в режимі реального часу.

Для запуску `hping3` з максимальної інтенсивністю було використано наступну команду, запущену з зовнішньої мережі:

```
sudo hping3 -S -flood -V -p 8080 192.168.2.1 --rand-source
```

Для запуску UDP-флуду високої інтенсивності, параметр «швидкість» було виставлено на максимум через графічний інтерфейс користувача. Також, для запуску `LOIC` у Linux середовищі необхідно використовувати засіб сумісності з ОС Windows – `vine`.

5.4 Експериментально отримані результати дослідження

Після проведення тестування було отримано наступні результати.

По-перше – аналізатор правильно виводить пакетну інформацію зі зразка protocols.pcap. Результати зображено на рис. 5.13.

```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
[sudo] password for alexandr:
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 3: udp_flood_analyzer ip_flood_analyzer network_analyzer
bridge #0(ext): pkt 4: ca:0:2b:7e:0:8 -> 0:50:56:e6:3e:ec ARP PTYPE 0x0800 2.2.2.9 is at ca:0:2b:7e:0:8
bridge #0(ext): pkt 3: 0:50:56:e6:3e:ec -> 0:0:0:0:0:0 ARP PTYPE 0x0800 Who has 2.2.2.9? Tell 2.2.2.1
bridge #0(ext): pkt 5: 10.1.2.3 -> 10.9.8.7 ID=1 ICMP Echo (ping) request id=0x0001, seq=1, ttl=64
bridge #0(ext): pkt 8: 10.9.8.7 -> 10.1.2.3 ID=2 ICMP Echo (ping) reply id=0x0004, seq=1, ttl=64
bridge #0(ext): pkt 6: 10.9.8.7 -> 10.1.2.3 ID=1 ICMP Echo (ping) reply id=0x0002, seq=1, ttl=64
bridge #0(ext): pkt 9: 10.10.10.1 -> 10.10.10.5 ID=1 TCP 100 -> 130 [SYN] Seq=1 Ack=0 Win=8192 Len=0
bridge #0(ext): pkt 7: 10.1.2.3 -> 10.9.8.7 ID=2 ICMP Echo (ping) request id=0x0003, seq=1, ttl=64
bridge #0(ext): pkt 12: 10.10.10.1 -> 10.10.10.5 ID=3 TCP 100 -> 130 [ACK] Seq=2 Ack=2 Win=8192 Len=8
bridge #0(ext): pkt 13: 10.10.10.1 -> 10.10.10.5 ID=4 TCP 100 -> 130 [ACK] Seq=10 Ack=2 Win=8192 Len=8
bridge #0(ext): pkt 11: 10.10.10.1 -> 10.10.10.5 ID=2 TCP 100 -> 130 [ACK] Seq=2 Ack=2 Win=8192 Len=0
bridge #0(ext): pkt 10: 10.10.10.5 -> 10.10.10.1 ID=1 TCP 130 -> 100 [SYN, ACK] Seq=1 Ack=2 Win=8192 Len=0
bridge #0(ext): pkt 15: 10.10.10.5 -> 10.10.10.1 ID=2 TCP 130 -> 100 [FIN, ACK] Seq=2 Ack=19 Win=8192 Len=0
bridge #0(ext): pkt 14: 10.10.10.1 -> 10.10.10.5 ID=5 TCP 100 -> 130 [FIN, ACK] Seq=18 Ack=2 Win=8192 Len=0
bridge #0(ext): pkt 16: 10.1.1.1 -> 10.1.1.2 ID=1 UDP 100 -> 200 Len=8
bridge #0(ext): pkt 17: 10.1.1.2 -> 10.1.1.1 ID=1 UDP 200 -> 100 Len=13
bridge #0(ext): pkt 19: 10.1.1.2 -> 10.1.1.1 ID=2 UDP 200 -> 100 Len=13
bridge #0(ext): pkt 18: 10.1.1.1 -> 10.1.1.2 ID=2 UDP 100 -> 200 Len=8
```

Рисунок 5.13 – Вивід пакетної інформації

По-друге – аналізатор не блокує легітимний трафік та не сприймає його за аномальний.

По-третє – аналізатор правильно виявляє DDoS атаку та UDP-флуд. Результати зображено на рис. 5.14.

```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 52757
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 40971
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 50314
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 58780
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 56214
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 37433
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 54806
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 42133
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 45137
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 46378
```

Рисунок 5.14 – Виявлення DDoS атаки та блокування UDP-флуду

Якщо казати про тестування під навантаженням, то маємо наступний результат – сервер не відповідає на запити під час DDoS атаки і аналізатор виявляє її присутність, а також – блокує UDP-флуд (див. рис. 5.15).

```
alexandr@alexand-laptop: ~/watchdog_install/bin
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 52757
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 40971
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 50314
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 50780
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 56214
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 37433
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 54806
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 42133
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 45137
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 46378

alexandr@alexand-laptop: ~/dev/dds_watchdog/env
File Edit View Search Terminal Help
alexandr@alexand-laptop:~/dev/dds_watchdog/env$ sudo ip netns exec external curl -X GET http://192.168.2.1:80/
curl: (6) Could not resolve host: 192.168.2.1

alexandr@alexand-laptop:~/dev/dds_watchdog/env$ sudo ip netns exec external hping3 -S --flood
V -p 80 192.168.2.1 --rand-source
using eth, addr: 192.168.1.1, MTU: 1500
HPING 192.168.2.1 (eth 192.168.2.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
... 192.168.2.1 hping statistic ...
20758413 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
alexandr@alexand-laptop:~/dev/dds_watchdog/env$ sudo ip netns exec external hping3 -S --flood
V -p 80 192.168.2.1 --rand-source
using eth, addr: 192.168.1.1, MTU: 1500
HPING 192.168.2.1 (eth 192.168.2.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
... 192.168.2.1 hping statistic ...
8597889 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
alexandr@alexand-laptop:~/dev/dds_watchdog/env$ sudo ip netns exec external hping3 -S --flood
V -p 80 192.168.2.1 --rand-source
using eth, addr: 192.168.1.1, MTU: 1500
HPING 192.168.2.1 (eth 192.168.2.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Рисунок 5.15 – Результати тестування під навантаженням

Також було перевірено вплив на пропускну здатність мережі під час використання аналізатора за допомогою утиліти ping [25]. Середня затримка складає 11.833 с. без здійснення аналізу та 12.495 с. зі здійсненням аналізу. Різниця складає 0.662 с. У відсотках це 5.57%. Рис. 5.16 ілюструє отримані результати.

```
alexandr@alexand-laptop: ~
File Edit View Search Terminal Help
alexandr@alexand-laptop:~$ sudo ip netns exec external ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=64 time=8.14 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=64 time=14.9 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=64 time=12.9 ms
64 bytes from 192.168.2.1: icmp_seq=4 ttl=64 time=10.9 ms
64 bytes from 192.168.2.1: icmp_seq=5 ttl=64 time=12.8 ms
64 bytes from 192.168.2.1: icmp_seq=6 ttl=64 time=10.9 ms
64 bytes from 192.168.2.1: icmp_seq=7 ttl=64 time=8.92 ms
64 bytes from 192.168.2.1: icmp_seq=8 ttl=64 time=14.8 ms
64 bytes from 192.168.2.1: icmp_seq=9 ttl=64 time=12.8 ms
64 bytes from 192.168.2.1: icmp_seq=10 ttl=64 time=10.9 ms
^C
--- 192.168.2.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9017ms
rtt min/avg/max/mdev = 8.146/11.833/14.902/2.160 ms
alexandr@alexand-laptop:~$ sudo ip netns exec external ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=64 time=8.60 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=64 time=14.9 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=64 time=12.9 ms
64 bytes from 192.168.2.1: icmp_seq=4 ttl=64 time=10.9 ms
64 bytes from 192.168.2.1: icmp_seq=5 ttl=64 time=9.87 ms
64 bytes from 192.168.2.1: icmp_seq=6 ttl=64 time=15.9 ms
64 bytes from 192.168.2.1: icmp_seq=7 ttl=64 time=13.9 ms
64 bytes from 192.168.2.1: icmp_seq=8 ttl=64 time=11.9 ms
64 bytes from 192.168.2.1: icmp_seq=9 ttl=64 time=9.93 ms
64 bytes from 192.168.2.1: icmp_seq=10 ttl=64 time=15.9 ms
^C
--- 192.168.2.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9016ms
rtt min/avg/max/mdev = 8.609/12.495/15.986/2.518 ms
```

Рисунок 5.16 – Вплив роботи аналізатора на мережну затримку

Також, під час стрес-тестування, жодних помилок та збоїв в роботі аналізатора не було виявлено.

Зміну значення інформаційної ентропії з плином часу під час тестування проілюстровано графіком на рис. 5.17. Початкова точка відповідає відсутності будь-якого мережного трафіку. На проміжку від 0 до 10 секунд з'являється легітимний трафік. На проміжку від 10 до 20 секунд цей трафік є насиченим, але все ще легітимним. Далі, починаючи з 20-ї секунди, починається DDoS атака й значення ентропії вже перевищує порогове значення 2.0. Пік знаходиться на проміжку від 40 до 60 секунд. Це час, коли атака досягла своєї максимальної потужності і майже кожен пакет з усієї кількості зчитаних пакетів на сталому інтервалі часу має унікальну IP-адресу (тобто, відправлений з окремої робочої станції). Тому значення ентропії сягає свого максимуму на цьому відрізку. Так як тестування проводилося із значенням часового інтервалу 2.0 секунди й довжиною вектору критичних значень у 15 елементів, то аналізатор вивів повідомлення про атаку приблизно через 30 секунд від її початку. Початком є час, коли значення ентропії перевищило порогове значення 2.0. Тобто, повідомлення про атаку було

виведено приблизно на 50-й секунді експерименту. Атака припиняється починаючи з 60-ї секунди, що й видно на графіку зі зниженням значення ентропії. На проміжку від 60 до 80 секунд маємо легітимний насичений трафік, який повністю припиняється на проміжку від 80 до 90 секунд. Аналіз завершено.

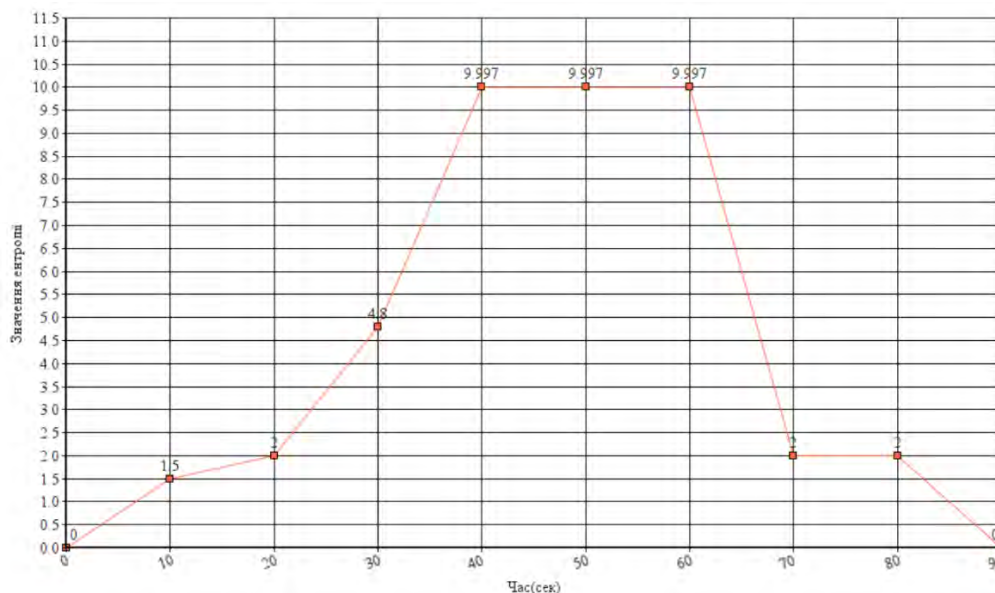


Рисунок 5.17 – Графік зміни значення інформаційної ентропії з плином часу

5.5 Аналіз, узагальнення та обґрунтування достовірності отриманих результатів

Проаналізувавши результати дослідження, викладені у попередньому розділі можна зробити наступні висновки.

Функціональне тестування:

- аналізатор правильно виводить пакетну інформацію;
- аналізатор не блокує легітимний трафік та не сприймає його за аномальний;
- аналізатор правильно виявляє DDoS атаку та UDP-флуд.

Стрес-тестування:

- аналізатор правильно та ефективно функціонує під великим навантаженням;
- вплив роботи аналізатора на пропускну здатність мережі складає приблизно 5-6%.

Базуючись на цих висновках можна сказати, що розроблене програмне забезпечення повністю відповідає поставленій задачі, працює ефективно та правильно. DDoS атака може бути ефективно виявлена аналізатором, що підтверджує правильність реалізації теоретично розроблених алгоритмів та самих алгоритмів. Вплив на пропускну здатність мережі знаходиться у межах 5-6 відсотків, що є хорошим показником, адже деякі IPS/IDS призводять до втрат у розмірі 15-20 відсотків під час найгірших сценаріїв використання. Це підтверджує ефективність та правильність реалізації багатопоточної архітектури.

Важливим є аналіз результатів зміни значень інформаційної ентропії з плином часу, графік якої зображено на рис. 5.17 з попереднього розділу. Отриманий графік повністю підтверджує теоретичну основу алгоритму виявлення, а отже й підтверджує правильність його реалізації, а також, що рішення стосовно його покращення не вплинуло на правильність роботи алгоритму, а тільки підвищило загальну ефективність всієї системи.

Графік виявлення UDP-флуду не був зроблений по причині відсутності сенсу його створення, адже цей алгоритм виявлення спрацьовує одразу при досягненні порогового значення кількості послідовних UDP-пакетів з однаковими значеннями контрольних сум у межах одного мережного потоку. Тому сам факт його спрацьовування, підтверджений результатами тестування, в тому числі й стрес-тестуванням, вже доводить твердження стосовно правильності теоретичного обґрунтування та реалізації цього алгоритму.

Звичайно, векторів атаки у складі DDoS ще існує доволі багато. Розроблена система покриває лише основні, найімовірніші сценарії перебігу атаки. Цього достатньо для більшості випадків, але підтримка більшої кількості потенційних

напрямків атаки є наступним етапом покращення та вдосконалення системи. Проте, результати цієї роботи доводять, що виявлення атак типу DDoS цілком можливе та може бути ефективним. Розроблене програмне забезпечення вже можна використовувати. Треба покращити модель керування процесом та логіку конфігурації, можливо також додати графічний інтерфейс користувача. Проте це вже питання подальшого вдосконалення системи.

Алгоритм виявлення UDP-флуду демонструє новий напрямок аналізу виявлення атак типу «флуд».

Подальші дослідження в цій області, як і покращення та вдосконалення розробленої системи, є дуже перспективними. Адже, як було згадано у вступі до цієї роботи, кількість активних Інтернет-користувачів щорічно зростає, як і залежність від постійної та злагодженої роботи мережних ресурсів. Тому питання ефективного виявлення та захисту від атак типу DDoS ставатиме ще більш актуальним у майбутньому.

6 ЕКОНОМІКО-ОРГАНІЗАЦІЙНА ЧАСТИНА

6.1 Концепція економічного обґрунтування

Метою даної роботи є розробка системи виявлення мережних атак типу DDoS, що дозволить кінцевим користувачам – власникам онлайн бізнесів – захистити свої сервіси від негативного впливу атак цього типу, що знизить потенційні збитки, спричинені відмовою в обслуговуванні клієнтів онлайн-сервісами.

Кількість активних користувачів мережі Інтернет щороку зростає, як і залежність від постійного та швидкого доступу до віддалених сервісів – банків, державних структур, джерел інформації, хмарних обчислень, тощо. Вже зараз більшість фінансових операцій здійснюється в електронному вигляді. Відмова в обслуговуванні на довгий час, наприклад, банківського сервісу, може завдати величезних збитків як компанії-власнику сервісу так і їхнім клієнтам.

Згідно з даними аналізу компанії Prolexis corp. – великого постачальника систем кібербезпеки, лише за останній 2019 рік було зафіксовано понад 7000 таких атак різного масштабу, що загалом призвело до збитків у розмірі понад 1 мільярд доларів США. Тому актуальність вирішуваної проблеми не тільки не зменшується, а й буде зростати у майбутньому.

Розроблений програмний продукт дозволяє встановити захист на стороні сервісу клієнта та своєчасно виявляти і блокувати DDoS атаки, знижуючи тим самим їх негативний вплив й зменшуючи об'єм фінансових втрат.

Програма є простою та ефективною у використанні. Велика кількість підтримуваних платформ дозволяє використовувати її у складі широкого спектру можливих мережних архітектур клієнтів.

Підсумовуючи сказане, створено таблицю сфер застосування та вигоди кінцевого користувача (табл. 6.1).

Таблиця 6.1 – Напрямки застосування та вигода користувачів системи

Зміст ідеї	Напрямки застосування	Вигода для кінцевих користувачів
Розробка системи виявлення атак типу DDoS	1. Моніторинг мережі клієнта	Аналіз активності користувачів
	2. Виявлення DDoS атак	Своєчасна протидія атаці та мінімізація фінансових збитків
	3. Блокування DDoS атак	Своєчасна протидія атаці та мінімізація фінансових збитків

На ринку послуг у сфері кібербезпеки зараз склалася така ситуація, коли кількість потенційних клієнтів більша за кількість пропозицій від компаній, що постачають послуги або готові рішення користувачам. Особливо у сфері захисту від DDoS атак. Також, як було сказано раніше, кількість потенційних клієнтів у майбутньому буде тільки зростати з ростом мережі Інтернет та темпів «діджиталізації». Конкурентів не багато, а для входу на ринок необхідні лише наступні речі: рекламна компанія, робота служби підтримки та підтримка робочого продукту командою розробників.

Підсумувати сказане можна за допомогою табл. 6.2.

Таблиця 6.2 – Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Головні конкуренти	CloudFlare
		Azure DDoS Protection
		Kaspersky DDoS Protection
2	Динаміка ринку (якісна оцінка)	Зростає

Продовження таблиці 6.2

№	Показники стану ринку	Характеристика
3	Наявність обмежень для входу	Якісна рекламна компанія

Головними стейкхолдерами є приватні або державні компанії, що займають сферу цифрового бізнесу – банки, розважальні платформи, державні замовники. Вони є потенційними інвесторами проекту.

Інтереси потенційних стейкхолдерів наступні:

- економічні інтереси;
- технологічні інтереси;
- репутація.

6.2 Визначення складу команди

Виходячи з аналізу обсягу та характеру виконання необхідних робіт, було визначено наступний склад команди:

- ініціатор проекту;
- команда розробників у розмірі трьох чоловік;
- тестувальник;
- бізнес-аналітик.

6.3 Планування етапів розробки продукту

Весь час розробки продукту можна поділити на 5 етапів:

- аналіз предметної області та вимог до розроблюваної системи;
- розробка системи;
- тестування розробленої системи та виправлення помилок;
- розробка рекламної кампанії;
- випуск готового продукту.

На повний цикл розробки – від початку аналізу предметної області до випуску готового продукту, відводиться шість місяців. Початок робіт – 12.05.2020, кінець робіт – 12.11.2020. Для кожного етапу вказується трудомісткість, кількість виконавців і тривалість робіт [26].

Тривалість робіт розраховується за формулою (6.1):

$$T_{\text{ц}} = \frac{Q}{R}, \quad (6.1)$$

де $T_{\text{ц}}$ – тривалість циклу, днів;

Q – трудомісткість, людино-днів;

R – кількість виконавців, людей.

Характеристика всіх етапів запланованих робіт зведена в табл. 6.3.

Таблиця 6.3 – Характеристика етапів запланованих робіт

Назва роботи	Трудомісткість		Виконавці	Тривалість, днів
	люд.-дні	% від підсумку		
1. Аналіз предметної області та вимог до розроблюваної системи	84	21%	Ініціатор, Команда розробників	21
2. Розробка системи	189	48%	Команда розробників	63
3. Тестування розробленої системи та виправлення помилок	84	21%	Тестувальник, Команда розробників	21
4. Розробка рекламної кампанії	10	3%	Бізнес-аналітик	10
5. Випуск готового продукту	30	7%	Команда розробників	10
Всього	397	100%		125

6.4 Визначення витрат на розробку

Для визначення витрат на розробку програми складається калькуляція кошторисної вартості робіт, що включає наступні статті розходів:

- основна заробітна плата;
- додаткова заробітна плата;
- відрахування на соціальні заходи;
- визначення матеріальних витрат та витрат по споживанню послуг;
- визначення витрат по амортизації;
- визначення інших операційних витрат.

6.4.1 Розрахунок основної заробітної плати

Витрати за цією категорією складаються з планового фонду зарплат всіх працівників, зайнятих у процесі виконання обсягу поставлених робіт. Розрахунок ведеться на основі даних трудомісткості та представлений у табл. 6.4.

Таблиця 6.4 – Розрахунок основної заробітної плати

Посада виконавця	Численність, люд.	Місячний оклад, грн.	Кількість місяців роботи	Сума ЗП, грн
Ініціатор	1	15000	1	15000
Розробник	3	15000	5.5	247500
Тестувальник	1	10000	1	10000
Бізнес-аналітик	1	10000	0,5	5000
Всього	6			277500

6.4.2 Розрахунок додаткової заробітної плати

Під додатковою заробітною платою приймається відсоток преміювання у розмірі 10% від основної заробітної плати, який розраховується за формулою (6.2):

$$ЗП_{\text{дод}} = ЗП_{\text{осн}} \cdot 0.1. \quad (6.2)$$

Підставивши величину основної заробітної плати, отримаємо наступне значення додаткової заробітної плати:

$$ЗП_{\text{дод}} = 277500 \cdot 0.1 = 27750 \text{ грн.}$$

6.4.3 Розрахунок відрахувань на соціальні заходи

Ця стаття розходів розраховується шляхом визначення суми єдиного соціального внеску. Згідно Податкового кодексу України, єдиний соціальний внесок сплачується роботодавцями у розмірі 22% від витрат на оплату праці. Розрахувати внесок можна за формулою (6.3):

$$ЄСВ = (ЗП_{\text{осн}} + ЗП_{\text{дод}}) \cdot 0.22. \quad (6.3)$$

Підставивши отримані попередньо дані зарплатних витрат, розрахуємо значення відрахувань на соціальні заходи:

$$ЄСВ = (277500 + 27750) \cdot 0.22 = 67155 \text{ грн.}$$

6.4.4 Розрахунок матеріальних витрат та витрат по споживанню послуг

Матеріальні витрати за весь час розробки проекту зведено до табл. 6.5.

Витрати по споживанню послуг за весь час розробки проекту зведено до табл. 6.6. Вартість електроенергії закладено на рівні споживання 300 кВт·год щомісяця.

Таблиця 6.5 – Матеріальні витрати

Стаття витрат	Кількість	Вартість за одиницю, грн.	Загальна вартість, грн.
1. Папір (формат А4)	2 упаковки	86	172
2. Картридж для принтера	1	230	230
3. Кулькова ручка	30	3	90
4. Блокнот (50 лист.)	6	100	600
Всього			1092

Таблиця 6.6 – Витрати по споживанню послуг

Назва послуги	Щомісячна вартість, грн.	Кількість місяців	Загальна вартість, грн.
1. Електроенергія	168	6	1008
2. Доступ до Інтернету	150	6	900
Всього			1908

6.4.5 Розрахунок витрат по амортизації

До цієї категорії відносяться витрати на придбання необхідного обладнання для розробки продукту [26]. В даному випадку – це закупівля комп'ютерів загальною вартістю 60000 грн.

Амортизаційні відчислення розраховуються за формулою (6.4):

$$A = BV_{03} \cdot H_A, \quad (6.4)$$

де A – річна сума амортизації, грн;

$BV_{OЗ}$ – балансова вартість основного засобу, грн;

H_A – річна норма амортизації, %.

При цьому, річна норма амортизації визначається за формулою (6.5):

$$H_A = \left(\frac{1}{T_{KB}} \right) \cdot 100, \quad (6.5)$$

де T_{KB} – термін корисного використання основного засобу, років.

Так як в нашому випадку термін виконання робіт складає шість місяці, що є менше одного року, а саме – 0.5 року, то амортизаційні відчислення розраховуються з урахуванням цього за формулою (6.6):

$$A = \frac{(BV_{OЗ} \cdot H_A)}{12} \cdot K_{MB}, \quad (6.6)$$

де K_{MB} – кількість місяців використання основного засобу.

Розрахуємо річну норму амортизації:

$$H_A = \frac{1}{0.5} \cdot 100 = 200\% = 2.$$

Балансова вартість основного засобу $BV_{OЗ}$ складає 60000 грн, кількість місяців використання K_{MB} складає 6 місяців. Розрахуємо амортизаційні відчислення:

$$A = \frac{60000 \cdot 2}{12} \cdot 6 = 60000 \text{ грн.}$$

6.4.6 Визначення інших операційних витрат

Операційні витрати визначимо у розмірі 10% від витрат на оплату праці за формулою (6.7):

$$ОВ = (ЗП_{\text{осн}} + ЗП_{\text{дод}}) \cdot 0.1. \quad (6.7)$$

Розрахуємо обсяги операційних витрат:

$$ОВ = (277500 + 27750) \cdot 0.1 = 30525 \text{ грн.}$$

6.4.7 Визначення сукупності операційних витрат

Сукупність усіх операційних витрат представляє собою виробничу собівартість продукту та складається з суми усіх витрат, визначених до цього. Результати визначення виробничої собівартості у вигляді калькуляції кошторисної вартості всіх робіт зведено до табл. 6.7.

Таблиця 6.7 – Калькуляція виробничої собівартості продукту

№	Найменування статей витрат	Сума, грн.	Питома вага від підсумку, %
1	Основна заробітна плата	277500	59,95
2	Додаткова заробітна плата	27750	5,99
3	Відрахування на соціальні заходи	64103	13,84
4	Матеріальні витрати та витрати по споживанню послуг	3000	0,67
5	Витрати по амортизації	60000	12,96
6	Інші операційні витрати	30525	6,59
7	Всього	462878	100,00

6.4.8 Визначення повної собівартості продукту

Повна собівартість продукту визначається шляхом складання сукупності операційних витрат (виробничої собівартості) та витрат на збут, та розраховується за формулою (6.8) [26]:

$$ПС = ВС + ВЗ, \quad (6.8)$$

де $ПС$ – повна собівартість продукту, грн.;

$ВС$ – виробнича собівартість продукту, грн.;

$ВЗ$ – витрати на збут, грн.

Так як для поширення продукту буде використовуватися цифрова дистрибуція, то витрати на збут складаються лише за кількістю витрат на рекламну кампанію у розмірі 20% від виробничої собівартості:

$$ВЗ = 462878 \cdot 0,2 = 92575,6 \text{ грн.}$$

Розрахуємо повну собівартість продукту:

$$ПС = 462878 + 92575,6 = 555453,6 \text{ грн.}$$

6.5 Визначення ціни реалізації

Визначимо ціну реалізації продукту, виходячи з повної собівартості, відсотка податку на додану вартість у розмірі 20%, та відсотка очікуваного прибутку у розмірі 25% за формулою (6.9):

$$Ц_p = ПС \cdot (1 + (ПДВ + ОП)), \quad (6.9)$$

де C_p – ціна реалізації, грн.;

$ПДВ$ – податок на додану вартість, %;

$ОП$ – очікуваний прибуток, %.

Розрахуємо ціну реалізації:

$$C_p = 555453,6 \cdot (1 + (0.2 + 0.25)) = 805407.72 \text{ грн.}$$

Базуючись на цінах конкурентів, встановимо ціну однієї копії програми на рівні 2000 грн:

$$C_k = 2000 \text{ грн.}$$

6.6 Визначення очікуваного доходу

Базуючись на аналізі потенційних покупців копій продукту, встановимо обсяг продажу копій програми у розмірі 1000 копій. Тоді, розрахуємо очікуваний дохід за формулою (6.10):

$$D = V \cdot C_k, \quad (6.10)$$

де D – очікуваний дохід, грн;

V – обсяг реалізації, копій програми;

C_k – ціна однієї копії програми, грн.

Обчислимо значення доходу від реалізації:

$$D = 1000 \cdot 2000 = 2000000 \text{ грн.}$$

6.7 Визначення фінансового результату

Фінансовим результатом є прибуток від реалізації розробленого продукту [27]. Прибуток розраховується за формулою (6.11):

$$\Pi = D - PC, \quad (6.11)$$

де Π – прибуток, грн;

D – дохід, грн;

PC – повна собівартість.

Обчислимо прибуток від реалізації розробленого продукту:

$$\Pi = 2000000 - 555453,6 = 1444546,4 \text{ грн.}$$

6.8 Висновки щодо економічної ефективності продукту

Розроблене програмне забезпечення, як продукт, дозволяє кінцевому користувачу захистити свою внутрішню мережу від загрози атак типу DDoS, що знижує фінансові втрати від атаки. Особливо це стосується представників Інтернет-бізнесу, для яких є дуже критичним, аби користувачі мали постійний доступ до сервісів. Адже відмова в обслуговуванні означає не лише великі фінансові збитки, а й удар по іміджевій складовій, по складовій довіри до компанії.

Економічна ефективність від використання програми полягає в прямому зниженні фінансових збитків від атак типу DDoS. Наприклад, якщо атака здійснюється на Інтернет-магазин в часи пікової активності клієнтів, то розрахувати економічну вигоду від використання продукту можна за формулою (6.12):

$$\Pi = T \cdot P_x, \quad (6.12)$$

де Π – це прибуток компанії внаслідок використання програми, грн.;

T – час здійснення атаки, хвилини;

Π_x – прибуток в хвилину, грн.

Якщо це великий інтернет-магазин, а час для атаки було обрано, наприклад, перед святами, то внаслідок її своєчасного виявлення розробленим програмним продуктом, можна розрахувати фінансову вигоду від використання наступним чином:

$$\Pi = 60 \cdot 100000 = 6000000 \text{ грн.}$$

Дані для обчислення результатів базуються на аналізі експертів стосовно прибутку великих інтернет-магазинів та торговельних площадок у святкові дні, а також, на даних дослідження методів здійснення DDoS атак і є обґрунтованими у першоджерелі.

Орієнтовний обсяг реалізації продукту на рік складає 100 копій. Тому можемо розрахувати річний дохід від реалізації продукту:

$$D_p = 100 \cdot C_k = 100 \cdot 2000 = 200000 \text{ грн.}$$

Тепер можемо розрахувати термін окупності:

$$T_o = \frac{PC}{D_p} = \frac{555453,6}{200000} = 2.78 \text{ року.}$$

А також, коефіцієнт ефективності капітальних вкладень:

$$E = \frac{1}{T_o} = \frac{1}{2.78} = 0.35,$$

що є швидшим, ніж в середньому серед продуктів-аналогів (середній термін окупності складає 3.5 – 4 роки, а коефіцієнт ефективності капітальних вкладень – 0.25), тому стейкхолдери зможуть повернути суму інвестицій раніше та почати отримувати прибуток [27].

Таким чином, можна зробити висновок, що розробка та впровадження програмного продукту є економічно доцільними як з точки зору сторони-ініціатора так й з точки зору стейкхолдерів-інвесторів.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Аналіз потенційних небезпек

Разом з впровадженням комп'ютеризації виникає потреба розуміти психологічні можливості людей, такі як емоційний стан, особливості пам'яті, швидкість реакції, увага, та ін. Через появу діяльності оператора суттєвих змін зазнала фахова структура праці. Зменшились ризики травматизму на виробництві, важкість фізичної праці, проте разом з цим, на людину посилюється вплив нових, раніше мало вивчених несприятливих виробничих факторів хімічного, фізичного і психофізіологічного впливу. Праця людини в умовах довготривалого статичного навантаження, великого нервово-емоційного напруження, зменшеної рухової активності, призводить до невротичних розладів, захворювань опорно-рухового апарату, серцево-судинної системи, органів зору, тощо. Телебачення, системи зв'язку, комп'ютери й інші засоби, є джерелом електромагнітного випромінювання, вплив якого на організм ще не до кінця вивчений.

Основа охорони праці в якості методології має науковий аналіз виробничого обладнання, технологічних процесів, робочих місць, умов праці, організації виробництва, трудових операцій, з метою виявлення небезпечних і шкідливих факторів виробництва, особливості та властивості їх впливу на організм.

В роботі досліджуються методи виявлення та протидії атакам типу DDoS та їх реалізація у вигляді готового програмного забезпечення. Команда розробників складається з трьох чоловік та одного тестувальника. Передбачається, що вони розміщуються в офісному приміщенні. Розміри приміщення наступні:

- довжина (A) – 8 метрів;
- ширина (B) – 5 метрів;
- висота (h) – 3,5 метри.

Площа приміщення становить:

$$S_{\text{п}} = 8 \cdot 5 = 40 \text{ м}^2.$$

Об'єм приміщення становить:

$$V_{\text{п}} = 8 \cdot 5 \cdot 3,5 = 140 \text{ м}^3.$$

Небезпечні та шкідливі виробничі фактори за природою виникнення поділяються на такі групи: фізичні (підвищена і знижена температура повітря, надмірна запиленість і загазованість повітря та ін.); хімічні (вміст у робочій зоні шкідливих речовин та ін.); психофізіологічні (нервово-емоційні перевантаження, розумове напруження та ін.); біологічні (підвищений вміст у повітрі мікроорганізмів та ін.).

У даному розділі було розглянуто наступні питання: склад повітря робочої зони та мікроклімат; освітлення робочого приміщення; шум та вібрації; виробниче випромінювання; безпека організації робочих місць; безпечність обладнання і процесу; електробезпека; пожежна безпека.

7.2 Заходи із забезпечення безпеки

Під час роботи в офісному приміщенні головним небезпечним фактором є електрика.

За ступенем небезпеки ураження електричним струмом, приміщення офісу відноситься до приміщень без підвищеної небезпеки.

Питання правильного електроживлення при підключенні комп'ютерів та їх об'єднанні в локальні мережі мають важливе значення як для забезпечення електробезпеки користувачів, так і для безаварійної та безперебійної роботи обладнання.

Заходи щодо забезпечення електробезпеки розроблено відповідно до ГОСТ 12.1.019– 90 ССБТ (Електробезпека. Загальні вимоги.) і ГОСТ 12.1.030– 91 ССБТ (Електробезпека. Захисне заземлення, занулення.).

Приміщення отримує електроенергію від центрального трансформатора, що знаходиться за його межами, і знижує напругу до значення 220 В, зі змінним струмом з частотою приблизно 50 Гц. Комп'ютер має блок живлення, який перетворює струм із мережі у постійний струм з напругою в 5 В або 12 В. Потужність блоків живлення становить від 300 до 450 Вт. Комп'ютери у приміщенні належать до I категорії захисту від ураження електрострумом, так як всі пристрої підключено до електромережі із заземленням, з використанням системи TN. Це така система, в якій електромережі мають глухе заземлення однієї з точок провідних частин джерела електроживлення, а частини електроустановки, що мають відкриті провідникові частини а також електроприймачі, під'єднані до цієї точки нейтральним та захисним провідниками. Всі електричні пристрої мають корпусі з ізолюючих матеріалів, а електроживлення здійснюється за допомогою спеціального кабелю, аби уникнути ураження електричним струмом людини. Визначені заходи з електробезпеки відповідають вимогам, запропонованим до приладів побутового вжитку у відповідності до міжнародного стандарту СЕ. Електропроводка у приміщенні з подвійною ізоляцією, чотирьохжильна та трифазна. Розетки встановлено на висоті 1 та 0,3 метри від рівня підлоги. Вимикачі розташовані на стінах на висоті 1,4 метра від рівня підлоги. Кабелі сховані у спеціальних коробах [29].

Електрична мережа вмикається і вимикається за допомогою пускової апаратури – блоку автоматів-запобіжників [29].

Для запобігання ураження людини струмом використано захисне заземлення. Сучасні комп'ютери вмикаються в розетку за допомогою триполюсної вилки, один з полюсів якої використовується для заземлення. З ним з'єднані металеві неструмоведучі частини комп'ютера, які можуть опинитися під напругою. В офісному приміщенні передбачається використання тільки триполюсних розеток, у яких відповідний полюс з'єднаний з шиною заземлення [29].

Кожен блок живлення комп'ютера або периферійного пристрою має мережевий фільтр. Конденсатори мережевого фільтра призначені для шунтування високочастотних перешкод живлячої мережі на землю, через дріт захисного заземлення [29].

При з'єднанні двох пристроїв кабелем, загальний провід інтерфейсів з'єднання пов'язаний із схемою заземлення та корпусом пристрою. Якщо пристрої, які з'єднуються, надійно заземлені через окремий провід на загальний контур, проблеми різниці потенціалів не виникає [29].

Згідно з "Правилами улаштування електроустановок" (ПУЕ), опір захисного заземлення за стандартом має становити не більше 4 Ом.

Згідно з ГОСТ 12.2.007.0–75 «Вироби електротехнічні. Загальні вимоги безпеки» в приміщенні встановлено автоматичні вимикачі, пристрій заземлення контуру.

7.3 Заходи з забезпечення виробничої санітарії та гігієни праці

Заходи щодо забезпечення виробничої санітарії та гігієни праці для офісного приміщення, обладнаного ПК з ВДТ, розроблені відповідно до вимог Державних санітарних норм та правил «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу», МІОУ 06.05.2014 р. за № 472/25249, ДСанПіН 3.3.2.007-98 «Державні стандартні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».

Робота програмістів в офісному приміщенні належить, за типом, до операторської роботи. Даний тип роботи відноситься до категорії 1а. Це робота, що виконуються сидячи і не потребує фізичного напруження. Робоче місце є постійним. Робота проводиться у теплий та холодний періоди року, тому варто підтримувати оптимальні за значенням мікрокліматичні показники.

Для забезпечення оптимального рівня параметрів повітряного середовища використано ДСН 3.3.6-042-99 «Санітарні норми мікроклімату виробничих приміщень» та ГОСТ 12.1.005-88 (1991) «ССБТ. Общие санитарно-гигиенические требования к воздуху рабочей зоны» (табл. 7.1).

Таблиця 7.1 – Норми мікроклімату в приміщенні

Період року	Температура, °С	Відносна вологість, %	Швидкість руху, м/с
Холодний	22-24	40-60	0,1
Теплий	23-25	40-60	0,1-0,2

В офісному приміщенні передбачено устрій системи водяного опалення приміщення для забезпечення необхідної температури повітря в холодний період року відповідно до ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування».

Проведені дослідження умов мікроклімату на робочих місцях з комп'ютерами показали, що взимку, значення відносної вологості повітря часто буває нижчими за встановлені норми й становить в середньому 25–35%. Це призводить до пересихання слизових оболонок носа, горла та очей, а також – до накопичення статичного електричного заряду, що виникає під час роботи комп'ютера. Для усунення цих проблем, в приміщенні передбачена наявність трьох зволожувачів повітря, які дозволяють підняти вологість в приміщенні до 60%. Кількість зволожувачів зумовлена тим, що один такий апарат розраховано на приміщення площею до 15 м².

У теплий період року для підтримки нормальної температури повітря в приміщенні використовуються кондиціонери типу «спліт-система». Два кондиціонери, розташовані у протилежних частинах офісного приміщення, дозволяють підтримувати температуру повітря не вище 24°C. Кількість кондиціонерів зумовлена тим, що один такий пристрій розраховано на кондиціонування приміщення площею до 20 м².

Вимоги до освітлення регламентовані ДБН В.2.5-28:2018, згідно з якими, освітлення у приміщеннях з комп'ютерами є суміщеним, за якого природне освітлення, що за нормами є недостатнім, має доповнюватися штучним. Природне освітлення одностороннє, бічне.

В офісному приміщенні коефіцієнт природної освітленості (КПО) повинен становити в середньому 1,1% (1,2% – у сонячний день та 1% – у хмарну погоду), що відповідає до вимогам ДБН В.2.5-28:2018. Для забезпечення відносно постійного природного освітлення всі вікна обладнанні металевими сонцезахисними жалюзі з можливістю регулювання. В жалюзі такого типу коефіцієнт відбиття становить 0,7. Робочі місця розміщено таким чином, аби у поле зору працівника не потрапляли світлі поверхні світильників або вікна [28].

Найбільш прийнятними для офісних приміщень є люмінесцентні лампи ЛБ (білого світла) та ЛТБ (тепло-білого світла) потужністю 20, 40 або 80 Вт [28].

Розрахуємо кількість світильників, необхідну для освітлення приміщення розміром 8 x 5 x 3,5 та освітленістю $E_n = 400$ лк. Для освітлення використовуються люмінесцентні лампи типу ЛБ в світильниках ЛПО. Коефіцієнт відбиття стелі становить 70%, а стін – 50% [28].

Розрахуємо індекс приміщення за формулою (7.1) [28]:

$$i = \frac{A \cdot B}{(h - 0.7) \cdot (A + B)} = \frac{8 \cdot 5}{(3.5 - 0.7) \cdot (8 + 5)} = 1.1. \quad (7.1)$$

Приймаємо коефіцієнт запасу $k = 1.6$ та коефіцієнт нерівномірності освітлення $Z = \frac{E_{\text{ср}}}{E_{\text{мін}}} = 1.1$, так як використовуються люмінесцентні лампи [28].

При індексі приміщення $i = 1.1$, отримаємо коефіцієнт використання $\eta = 40\%$ [28].

Світильники розміщено в два ряди: $N_p = 2$.

Визначаємо необхідний світловий потік ламп в кожному ряду за формулою (7.2) [28]:

$$\Phi_p = \frac{E_n \cdot S \cdot Z \cdot k}{N_p \cdot \eta} = \frac{400 \cdot 40 \cdot 1.1 \cdot 1.6}{2 \cdot 0.4} = 35200 \text{ лм.} \quad (7.2)$$

Якщо в світильнику встановлено по дві лампи ЛБ ($n = 2$) потужністю 80 Вт та світловим потоком $\Phi_{\text{л}} = 5200$ лм, то необхідна кількість світильників в ряду розраховується за формулою (7.3) [28]:

$$N_{\text{св.р}} = \frac{\Phi_p}{n \cdot \Phi_{\text{л}}} = \frac{35200}{2 \cdot 5200} = 3.38 \approx 4. \quad (7.3)$$

Тоді розрахуємо необхідну загальну кількість світильників за формулою (7.4):

$$N = N_p \cdot N_{\text{св.р}} = 2 \cdot 4 = 8. \quad (7.4)$$

Для виключення ситуацій засвічення екранів дисплеїв прямими світловими потоками від світильників, їх розміщують збоку від робочого місця, паралельно лінії зору оператора. Так само, паралельно лінії зору оператора, мають бути стіни з вікнами, для запобігання засвічення екранів дисплеїв природним світлом [28].

Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється стомленість у зв'язку з підвищеними енергетичними витратами та нервово-психічною напругою.

Згідно з технічним паспортом обладнання (комп'ютери, монітори, принтери, клавіатури (при натисканні), маніпулятори типу «миша» (при натисканні), «тачпад» (при натисканні)), що використовується для роботи, рівень шуму складає: 43 дБА. Згідно з ДСН 3.3.6-039-99, нормальне значення рівня шуму є більшим за фактичне значення, тому додаткових робіт для усунення надмірного

шуму не проводилося. Робота в даному офісному приміщенні з перерахованим обладнанням відповідає нормативним вимогам щодо шуму та вібрації.

На оператора комп'ютера, в період роботи за дисплеєм (екраном монітора), здійснюється електромагнітне випромінювання високої інтенсивності, яке може призвести до професійних захворювань. Комп'ютери відносять до 5-го діапазону (низькі частоти) [28].

Основним джерелом електромагнітного випромінювання в офісному приміщенні, у тому числі рентгенівського, є електронно-променеві трубки (ЕПТ) старих моніторів. Тому, для зниження потужності експозиційної дози рентгенівського випромінювання для роботи використовуються монітори з рідкокристалічними дисплеями [28].

Обладнання і організація робочого місця в офісному приміщенні мають забезпечувати відповідність конструкції кожного з елементів робочого місця та їхнього розташування вимогам ергономіки, враховуючи характер та особливості трудової діяльності [28].

Площа на одне робоче місце складає 8 м^2 , а об'єм – 28 м^3 . Приміщення має природне та штучне освітлення. Робочі місця розташовано так, щоб природне світло падало збоку [28].

Робочі місця та їх взаємне розташування відповідає вимогам антропометрії, психологічним та фізичним вимогам.

Робочі стільці мають механізми підйому і повороту, та регулюються по висоті і куту нахилу сидіння й спинки, а також – по відстані між спинкою та переднім краєм сидіння. Регулювання кожного параметра незалежне та здійснюється легко, має надійну фіксацію. Екран монітору знаходиться на оптимальній відстані – 700 мм [28].

Робота за комп'ютером характеризується великою розумовою та нервово-емоційною напругою, значною напруженістю зору, та доволі великим навантаженням на м'язи рук під час роботи з комп'ютерною клавіатурою. Велике значення має правильне розташування та конструкція компонентів робочого місця, що дуже важливо для підтримки оптимальної робочої пози оператора [28].

В офісному приміщенні наявні дві медичні аптечки першої допомоги.

Також, приміщення обладнано полицями, шафами, тумбами й стелажками для зберігання документів та розміщення інших носіїв інформації.

Колір приміщень і меблів сприяє створенню комфортних умов для зорового сприйняття.

7.4 Заходи з забезпечення безпеки у надзвичайних ситуаціях

7.4.1 Заходи з пожежної безпеки

Пожежа в офісному приміщенні становить особливу небезпеку, тому що пов'язана з підвищеною загрозою для життя та здоров'я людей через замкнутість та відносно невеликий об'єм приміщення, а також – зі значними матеріальними втратами. Як відомо, пожежа може виникнути при взаємодії горючих речовин, окислення і джерел запалювання. У приміщеннях офісу присутні всі три основні чинники, необхідні для виникнення пожежі [30].

Для відводу теплоти від комп'ютерів в приміщенні постійно діє потужна система кондиціонування. Тому кисень, як окиснювач процесів горіння, є всюди.

Приміщення офісу, згідно з ДСТУ Б В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх установок за вибуховою та пожежною небезпекою» відноситься до категорії «П-Па», а клас можливої пожежі, згідно з ДБН В.1.1-7:2016 «Пожежна безпека об'єктів будівництва. Загальні вимоги», визначається як «А».

До засобів гасіння пожежі, призначених для локалізації невеликих загорянь, відносяться пожежні стовбури, внутрішні пожежні водопроводи, вогнегасники, сухий пісок, азбестові ковдри і т. п [30].

Оскільки приміщення, що обладнане комп'ютерами з ВДТ, має площу 40 м², відповідно до вимог п. 3.8 розділу «Типові норми належності вогнегасників» ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників. Загальні технічні вимоги» для гасіння електроустановок, що знаходяться під напругою, передбачені вуглекислотні вогнегасники типу ВВК-3,5 у кількості 2

штук (з розрахунку один вогнегасник з величиною заряду вогнегасної речовини 3 кг і більше, на 20 м² площі приміщення). Відстань між вогнегасниками та місцями можливих загорянь не перевищує 10 м [30].

Усі працівники офісу під час прийняття на роботу й в процесі праці, повинні проходити інструктаж з пожежної безпеки та перевірку знань з протипожежної безпеки. Евакуаційні виходи й проходи утримуються постійно вільними та нічим не захащуються [30].

Відповідальний за протипожежний стан офісного приміщення зобов'язаний періодично (не рідше одного разу на місяць) перевіряти на працездатність пристрій, який забезпечує автоматичне відключення вентиляційної системи у разі виникнення пожежі, а також пристроїв для вогнедимозатримувальня. Варто не допускати руйнування або порушення цілісності під час ремонту негорючих діафрагм у просторі під підлогою [30].

Фільтри для вентиляції припливно-витяжного типу треба очищати згідно із завчасно затвердженим графіком, але не рідше ніж два рази на рік. Роботи з ремонту вузлів (блоків) ПК, проводити не дозволяється. Їх необхідно здійснювати в окремому, спеціально обладнаному, приміщенні. У разі необхідності проведення технічного обслуговування чи ремонту ПК власне в приміщенні, допускається мати не більше ніж 0,6 л легкозаймистої рідини у небиткій та закритій тарі. Для промивання деталей необхідно застосовувати, за можливості, миючі препарати що не є горючими. Будь які інші знімні пристрої можна промивати за допомогою рідин, що є горючими, проте тільки в спеціальному приміщенні яке, обладнане вентиляцією [30].

Відповідальний за пожежну безпеку офісного приміщення зобов'язаний негайно повідомити службу технічної підтримки у разі появи незадовільного технічного стану теплових та димових сповіщувачів автоматичної протипожежної сигналізації, а також – про несправність автоматичної установки газового пожежогасіння [30].

Усі працівники офісу повинні вміти привести в дію первинні засоби гасіння пожежі. Регулярно, але не рідше ніж один раз на квартал, варто очищати від пилу

ПК й іншу апаратуру, кабельні траншеї та підлоговий простір за допомогою пиłosоса [30].

У межах офісного приміщення забороняється:

- залишати без нагляду увімкнену електричну апаратуру;
- зберігати постійно носії інформації, запасні блоки та деталі в кількості більшій, ніж необхідно для поточного використання;
- користуватися груповими розетками на горючій рanelі;
- використовувати доріжки та килими, зроблені із синтетичних матеріалів;
- ставити ґрати на вікна;
- застосовувати електронагрівальні побутові прилади;
- курити та застосовувати відкритий вогонь (газо-електрозварювальні роботи проводяться за наявності спеціального дозволу адміністрації у встановленому порядку) [30].

Після закінчення робочого часу всі робочі місця мають бути прибрані, з приміщення необхідно видалити горючі відходи та вимкнути всі струмоприймачі [30].

У разі виявлення ознак пожежі працівник повинен:

- негайно повідомити службу порятунку за номером телефону – 101 (112). При цьому необхідно зазначити адресу об'єкта, місце де виникла пожежа, кількість поверхів, поточний стан, наявність людей, своє прізвище;
- вжити заходів щодо евакуації людей та матеріальних цінностей;
- вжити заходів щодо гасіння пожежі наявними засобами пожежогасіння [30].

Керівник підрозділу, якого повідомили про пожежу, зобов'язаний:

- перевірити, чи викликані пожежно-рятувальні підрозділи, проінформувати про пожежу керівництво;
- вимкнути, у разі необхідності, струмоприймачі та вентиляцію;

- за умови загрози життю та здоров'ю людей, негайно почати їх евакуацію, вивести за межі зони небезпеки всіх осіб, які не пов'язані із ліквідацією пожежі;
- перевірити оповіщення людей про пожежу;
- забезпечити дотримання техніки безпеки для всіх працівників, котрі беруть участь у пожежогасінні;
- організувати зустріч загонів Державної пожежної безпеки, надати їм допомогу під час заходів з ліквідації пожежі;
- після прибуття пожежно-рятувальних підрозділів, забезпечити їм вільний доступ до місця появи пожежі [30].

7.4.2 Заходи з цивільного захисту

Розглянуто питання порядку дій сил цивільного захисту у вогнищі хімічного ураження.

Під час виникнення аварій на хімічно небезпечних об'єктах можливі:

- залпові викиди небезпечних хімічних речовин у довкілля;
- пожежі з виділенням токсичних речовин;
- забруднення об'єктів і місцевості в осередках аварії та на сліді розповсюдження хмари;
- широкі зони задимлення в сполуці з токсичними продуктами [31].

Під час аварії можуть діяти, як правило, декілька факторів ураження: пожежа, вибухи, хімічне забруднення повітря і місцевості, а за межами об'єкта - забруднення довкілля. Аварії на хімічно небезпечних об'єктах характеризуються високою швидкістю формування і дією вражаючих факторів. У зв'язку з цим, заходи щодо захисту особового складу та населення, локалізації та ліквідації наслідків аварії повинні проводитись у мінімально можливі строки [31].

Керівник рятувальними роботами, при надходженні даних про виникнення хімічної аварії, оцінює масштаби можливого зараження і визначає кількість

населення, що проживає в районах, яким загрожує небезпека при поширенні хмари зараженого повітря, ставить завдання підрозділам хімічної та медичної розвідки, віддає необхідні розпорядження про проведення заходів щодо захисту населення і організує проведення рятувальних робіт в осередках ураження [31]. На підставі оцінки ситуації приймає рішення, в якому вказує:

- райони і наслідки руйнувань хімічно небезпечних об'єктів;
- обсяг рятувальних робіт і послідовність їх проведення;
- сили та засоби, які залучаються для виконання рятувальних та інших невідкладних робіт та їх завдання;
- напрямки (ділянки, об'єкти) зосередження основних зусиль;
- заходи безпеки при виконанні рятувальних робіт;
- маршрути та місця евакуації потерпілих і населення з зон хімічного зараження;
- порядок взаємодії та заходи щодо забезпечення дій сил ЦЗ;
- склад і район розташування резерву;
- порядок організації управління [31].

Для проведення рятувальних робіт, в залежності від типу аварії, залучаються невоєнізовані формування об'єктів господарчої діяльності, територіальних органів, підрозділи цивільного захисту, а також, при необхідності, військові частини і підрозділи МО, МВС. Розпорядження на виконання конкретних завдань в осередку хімічного ураження командири формувань та підрозділів отримують від керівника рятувальними роботами, в оперативному підпорядкуванні якого вони перебувають [31].

Отримавши завдання, командир зобов'язаний:

- усвідомити завдання;
- зорієнтувати підлеглих на майбутні дії;
- визначити заходи, які необхідно провести негайно, для забезпечення повної готовності до виконання отриманої завдання;
- провести розрахунок часу;

- дати вказівку командирам (начальникам) груп про підготовку своїх підрозділів до виконання завдання, про організацію хімічної розвідки, про час і порядок роботи на місцевості;
- оцінити обстановку;
- прийняти рішення на проведення рятувальних робіт, доповісти його безпосередньому начальнику і довести до заступників та командирів підпорядкованих йому підрозділів;
- організувати роботу на місцевості;
- дати розпорядження на виконання рятувальних робіт;
- організувати взаємодію та всебічне забезпечення виконання робіт і управління [31].

При виявленні масштабів осередку хімічного зараження аналізуються:

- райони (об'єкти), піддані хімічному зараженню;
- вид НХР;
- метеорологічні умови і їх вплив на поширення зараженого повітря, глибину зони зараження;
- тривалість впливу НХР;
- можливі ділянки (об'єкти) затікання і застою НХР;
- характер можливих руйнувань, місця пожеж та уражень в районі та на ділянці (об'єкті) рятувальних робіт, на шляхах висування;
- види майбутніх рятувальних робіт та їх орієнтовний обсяг [31].

При оцінці району виконання рятувальних робіт, аналізується характер місцевості та її вплив на дії підрозділів при виконанні поставлених завдань у засобах індивідуального захисту, погодні умови, що визначають час безперервного перебування особового складу в засобах індивідуального захисту [31].

У результаті з'ясування завдання, оцінки обстановки і проведених розрахунків, командир приймає рішення і віддає розпорядження на виконання завдань [31].

Рятувальні роботи в осередках хімічного ураження виконуються в протигазах і засобах захисту шкіри. Тривалість роботи особового складу однієї зміни в осередку хімічного ураження залежить від часу допустимого безперервного перебування в засобах індивідуального захисту. Залежно від обстановки, роботи в осередках хімічного ураження можуть проводитися по черзі, тобто – в окремих найбільш небезпечних ділянках, або одночасно – одразу на всій території [31].

Осередки хімічного ураження вважаються ліквідованими, коли перебування людей без засобів захисту в них стає безпечним [31].

ВИСНОВКИ

В результаті виконання роботи було досліджено, покращено та імплементовано алгоритм виявлення присутності атак типу DDoS на основі розрахунку інформаційної ентропії розподілу мережних пакетів на сталих інтервалах часу. Розроблено, теоретично обґрунтовано та реалізовано алгоритм виявлення UDP флуду.

Реалізація має вигляд готового системного програмного забезпечення під Unix-подібні операційні системи, що здатне виявляти та блокувати атаки типу DDoS правильно та ефективно, з мінімальним впливом на пропускну здатність мережі.

Для роботи з мережею використано бібліотеку для C `librcap`. Сам аналізатор розроблено засобами мови програмування C++ для швидкодії. В основі виявлення аномалій мережного трафіку лежать два алгоритми. Перший базується на розрахунку ентропії Шеннона а другий – на аналізі контрольних сум послідовностей мережних пакетів.

Так як аналізатор розроблений з використанням виключно мови C++ та бібліотеки `librcap` для платформи Linux, є можливість його використання у складі серверу або мережевого обладнання на базі Unix-подібних операційних систем.

Науковою новизною роботи є оптимізація логіки алгоритму, що базується на розрахунку інформаційної ентропії, шляхом використання сталих інтервалів часу замість фіксованих пакетних інтервалів. Проведене експериментальне дослідження підтвердило правильність та ефективність роботи вдосконаленого алгоритму, який може бути використаний для успішного виявлення аномалій мережного трафіку без впливу на легітимний трафік та з мінімальною мережною затримкою.

Також, науковою новизною є розроблений алгоритм виявлення UDP-флуду на основі аналізу значень контрольних сум послідовних UDP-пакетів. За результатами проведеного експериментального дослідження розроблений

алгоритм показав свою ефективність та правильність роботи без впливу на легітимний UDP-трафік та з мінімальною мережною затримкою.

Результати роботи можуть бути використані для подальшого дослідження предметної області, оптимізації розроблених алгоритмів та для розробки нових методів виявлення атак типу DDoS. На практиці, програмне забезпечення може бути покращене додаванням графічного інтерфейсу користувача, системи гнучкого віддаленого управління та відправки нотифікацій, тощо.

Розроблений програмний засіб можна використовувати як компонент захисту мережі від DDoS атак на широкому спектрі серверів та мережного обладнання.

ПЕРЕЛІК ПОСИЛАНЬ

1. DoS-атака [Електронний ресурс] . – Режим доступу: <https://uk.wikipedia.org/wiki/DoS-атака>.
2. Аналізатор трафіку [Електронний ресурс] . – Режим доступу: https://uk.wikipedia.org/wiki/Аналізатор_трафіку.
3. Сетевой мост [Электронный ресурс] . – Режим доступа: https://ru.wikipedia.org/wiki/Сетевой_мост.
4. DDoS Survival Handbook : learning guide [Electronic resource] / Ronen Kenig, Deborah Manor, Ziv Gadot, Daniel Trauner. – Radware, Ltd., 2013. – 54 с.
5. User Datagram Protocol [Electronic resource] . – Access mode: <https://tools.ietf.org/html/rfc768>.
6. Internet Control Message Protocol [Electronic resource] . – Access mode: <https://tools.ietf.org/html/rfc792>.
7. Internet Group Management Protocol [Electronic resource] . – Access mode: <https://tools.ietf.org/html/rfc2236>.
8. Transmission Control Protocol [Electronic resource] . – Access mode: <https://tools.ietf.org/html/rfc793>.
9. LOIC GitHub repository [Electronic resource] . – Access mode: <https://github.com/NewEraCracker/LOIC/>.
10. hping3 [Electronic resource] . – Access mode: <https://linux.die.net/man/8/hping3>.
11. DDoS Attack Detection Algorithms Based on Entropy Computing : research article [Electronic resource] / Liying Li, Jianying Zhou, Ning Xiao. – Singapore : National University of Singapore, 2007. – 16 с.
12. std::unordered_map [Electronic resource] . – Access mode: https://en.cppreference.com/w/cpp/container/unordered_map.
13. Ланцюжок обов'язків [Електронний ресурс] . – Режим доступу: <https://refactoring.guru/uk/design-patterns/chain-of-responsibility>.

14. Мережевий інтерфейс [Електронний ресурс] . – Режим доступу: https://uk.wikipedia.org/wiki/Мережевий_інтерфейс.
15. DFD [Електронний ресурс] . – Режим доступу: <https://ru.wikipedia.org/wiki/DFD>.
16. Component diagram [Electronic resource] . – Access mode: https://en.wikipedia.org/wiki/Component_diagram.
17. C++ Language Basic Concepts [Electronic resource] . – Access mode: https://en.cppreference.com/w/cpp/language/basic_concepts.
18. tcpdump [Electronic resource] . – Access mode: <http://www.tcpdump.org/>.
19. CMake [Electronic resource] . – Access mode: <https://cmake.org/>.
20. Class Diagrams [Electronic resource] . – Access mode: <https://www.uml-diagrams.org/class-diagrams.html>.
21. Thread support library [Electronic resource] . – Access mode: <https://en.cppreference.com/w/cpp/thread>.
22. UML Sequence Diagrams [Electronic resource] . – Access mode: <https://www.uml-diagrams.org/sequence-diagrams.html>.
23. ip [Electronic resource] . – Access mode: <https://linux.die.net/man/8/ip>.
24. tcpreplay [Electronic resource] . – Access mode: <https://linux.die.net/man/1/tcpreplay>.
25. ping [Electronic resource] . – Access mode: <https://linux.die.net/man/8/ping>.
26. Бойчик І.М. Економіка підприємств : навч. посібник / І. М. Бойчик, П. С. Харів, М. І. Хопчан. – Львів: В-во «Сполом». – 2009. – 212 с.
27. Вовчак О. Д. Інвестування : навч. посіб. / О. Д. Вовчак. - Л. : Новий Світ – 2000, 2008. – 544 с.
28. Методичні вказівки до лабораторно заняття «Дослідження загального освітлення виробничих приміщень» з дисципліни «Цивільний захист і охорона праці в галузі» для студентів всіх форм навчання / Укл. В.І. Шмирко, О.В. Коробко, Ю.І. Троян. – Запоріжжя: ЗНТУ. Каф. ОПіНС, 2020. – 34 с.

29. Методичні вказівки до лабораторного заняття «Дослідження захисного заземлення та методика його розрахунку» з дисципліни «Цивільний захист і охорона праці в галузі» для студентів усіх спеціальностей та усіх форм навчання / Укл. М.О. Журавель, С.М. Журавель. – Запоріжжя: ЗНТУ. Каф. ОПіНС, 2017. – 30 с.

30. Методичні вказівки до лабораторного заняття «Дослідження властивостей первинних засобів пожежогасіння, визначення їх типів та розрахунок кількості» з дисципліни «Цивільний захист і охорона праці в галузі» для студентів технічного напрямку, усіх форм навчання / Укл. А.С. Петрищев, С.М. Журавель. – Запоріжжя: ЗНТУ. Каф. ОПіНС, 2017. – 26 с.

31. Цивільна оборона : навч. посіб. для студ. вищ. вищ. навч. закл. / [Кулаков М.А. та ін.] ; за ред. проф. В.В. Березуцького. – Х.: Факт, 2008. – 312 с.

ДОДАТОК А
Текст програми

Вихідний код програми розміщено на сервісі GitHub у публічному доступі (https://github.com/Normandez/ddos_watchdog), під власним авторством та ліцензією MIT:

MIT License

Copyright (c) 2020 Oleksandr Serhienko <sergienko.9711@gmail.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Вихідний код, розміщений у цьому додатку, відповідає останній ревізії контролю версій на момент складання додатку, а саме:

cacb1b7eb4db360527b2776b97f41646edef30b2

A.1 Текст програми модулю core

A.1.1 Файл config.h

```
#ifndef CONFIG_H
#define CONFIG_H

#include <string>

struct Config
{
public:
    Config() = default;
    Config(const Config&) = delete;
    Config& operator=(const Config&) = delete;

    static Config* get_instance()
    {
```

```

        static Config conf;
        return &conf;
    }

public:
    bool load_config(const char* file_name);

public:
    std::string int_iface;
    std::string ext_iface;
    std::string detectors;

    // network_analyzer config
    int max_print_threads = 0;

    // ip_flood_analyzer config
    int analyze_time_window_ip = 0;           // in seconds
    short threshold_vector_size = 0;
    float entropy_threshold = 0.0;

    // udp_flood_analyzer config
    int analyze_time_window_udp = 0;          // in seconds
    long threshold_pkt_num = 0;
    std::string action_type = "";             // "alert"|"block"
};

#endif // CONFIG_H

```

A.1.2 Файл config.cc

```

#include "config.h"

#include <fstream>

#include "logger.h"

// Common config
static const std::string int_iface_conf_key = "INT_IFACE=";
static const std::string ext_iface_conf_key = "EXT_IFACE=";
static const std::string detectors_key = "DETECTORS=";

// NetworkAnalyzer config
static const std::string max_print_threads_key = "MAX_PRINT_THREADS=";

// IpFloodAnalyzer config
static const std::string analyze_time_window_ip_key = "ANALYZE_TIME_WINDOW_IP=";
static const std::string threshold_vector_size_key = "THRESHOLD_VECTOR_SIZE=";
static const std::string entropy_threshold_key = "ENTROPY_THRESHOLD=";

// UdpFloodAnalyzer config
static const std::string analyze_time_window_udp_key = "ANALYZE_TIME_WINDOW_UDP=";
static const std::string threshold_pkt_num_key = "THRESHOLD_PKT_NUM=";
static const std::string action_type_key = "ACTION_TYPE=";

bool Config::load_config(const char* file_name)
{
    std::ifstream file;
    file.open(file_name);

```

```

if ( !file.is_open() )
{
    Logger::error("can't open config file: %s", file_name);
    return false;
}

std::string buf = "";
while ( file >> buf )
{
    if( buf.find(int_iface_conf_key) != std::string::npos )
        int_iface = buf.substr(int_iface_conf_key.size());
    else if ( buf.find(ext_iface_conf_key) != std::string::npos )
        ext_iface = buf.substr(ext_iface_conf_key.size());
    else if ( buf.find(detectors_key) != std::string::npos )
        detectors = buf.substr(detectors_key.size());
    else if ( buf.find(max_print_threads_key) != std::string::npos )
        max_print_threads =
std::stoi(buf.substr(max_print_threads_key.size()));
    else if ( buf.find(analyze_time_window_ip_key) != std::string::npos )
        analyze_time_window_ip =
std::stoi(buf.substr(analyze_time_window_ip_key.size()));
    else if ( buf.find(threshold_vector_size_key) != std::string::npos )
        threshold_vector_size =
std::stoi(buf.substr(threshold_vector_size_key.size()));
    else if ( buf.find(entropy_threshold_key) != std::string::npos )
        entropy_threshold =
std::stof(buf.substr(entropy_threshold_key.size()));
    else if ( buf.find(analyze_time_window_udp_key) != std::string::npos )
        analyze_time_window_udp =
std::stoi(buf.substr(analyze_time_window_udp_key.size()));
    else if ( buf.find(threshold_pkt_num_key) != std::string::npos )
        threshold_pkt_num =
std::stol(buf.substr(threshold_pkt_num_key.size()));
    else if ( buf.find(action_type_key) != std::string::npos )
        action_type = buf.substr(action_type_key.size());
}

file.close();
Logger::log("config file loaded: %s", file_name);

return true;
}

```

A.1.3 Файл detector_manager.h

```

#ifndef DETECTOR_MANAGER_H
#define DETECTOR_MANAGER_H

#include <string>
#include <sys/types.h>

#include "net_defines.h"

class DetectorManager
{
public:
    static void init_pipeline(const size_t bridge_id, const PktDirection dir);
    static void cleanup_pipeline();

```

```

static bool execute(const u_char* pkt, const unsigned int pkt_len,
    const unsigned long long pkt_num);

static size_t get_pipeline_len();
static std::string get_pipeline_names();

};

#endif // DETECTOR_MANAGER_H

```

A.1.4 Файл detector_manager.cc

```

#include "detector_manager.h"

#include <list>

#include "detectors/detector.h"
#include "detectors/network_analyzer.h"
#include "detectors/ip_flood_analyzer.h"
#include "detectors/udp_flood_analyzer.h"

#include "config.h"

thread_local std::list<Detector*> s_detectors;

void DetectorManager::init_pipeline(const size_t bridge_id, const PktDirection
dir)
{
    Config* conf = Config::get_instance();

    if ( conf->detectors.find(udp_flood_analyzer_name) != std::string::npos )
    {
        s_detectors.push_back(
            new UdpFloodAnalyzer(bridge_id, dir, conf->analyze_time_window_udp,
                conf->threshold_pkt_num, conf->action_type));
    }

    if ( conf->detectors.find(ip_flood_analyzer_name) != std::string::npos )
    {
        s_detectors.push_back(
            new IpFloodAnalyzer(bridge_id, dir, conf->analyze_time_window_ip,
                conf->threshold_vector_size, conf->entropy_threshold));
    }

    if ( conf->detectors.find(network_analyzer_name) != std::string::npos )
        s_detectors.push_back(new NetworkAnalyzer(bridge_id, dir, conf-
>max_print_threads));
}

void DetectorManager::cleanup_pipeline()
{
    for ( auto detector : s_detectors )
        delete detector;
}

bool DetectorManager::execute(const u_char* pkt, const unsigned int pkt_len,
    const unsigned long long pkt_num)
{

```

```

    for ( auto& detector : s_detectors )
    {
        if ( !detector->analyze(pkt, pkt_len, pkt_num) )
            return false;
    }

    return true;
}

size_t DetectorManager::get_pipeline_len()
{ return s_detectors.size(); }

std::string DetectorManager::get_pipeline_names()
{
    std::string list_names = " ";
    for ( auto& detector : s_detectors )
    {
        std::string name = detector->get_name();
        name += " ";
        list_names.append(name);
    }

    return list_names;
}

```

A.1.5 Файл inline_set.h

```

#ifndef INLINE_SET_H
#define INLINE_SET_H

#include <atomic>
#include <thread>

class Sniffer;

class InlineSet
{
public:
    InlineSet(const char* ext_iface, const char* int_iface, size_t bridge_id,
              std::atomic_ullong& pkt_counter);

    virtual ~InlineSet();

    bool open();
    virtual void live() = 0;

protected:
    Sniffer* ext_sniffer = nullptr;
    Sniffer* int_sniffer = nullptr;
    size_t id;
};

class LiveBridge
{
public:
    LiveBridge(const char* ext_iface, const char* int_iface, int bridge_id);
    ~LiveBridge();
}

```

```

    bool open();
    void live();

private:
    InlineSet* ext_to_int = nullptr;
    InlineSet* int_to_ext = nullptr;
    size_t id;
    std::atomic_ullong pkt_counter;

    std::thread* etoi = nullptr;
    std::thread* itoe = nullptr;

};

#endif // INLINE_SET_H

```

A.1.6 Файл inline_set.cc

```

#include "inline_set.h"

#include "detector_manager.h"
#include "logger.h"
#include "sniffer.h"
#include "thread_context.h"

// Inline sets

InlineSet::InlineSet(const char* ext_iface, const char* int_iface, size_t
bridge_id,
    std::atomic_ullong& pkt_counter)
    : id(bridge_id)
{
    ext_sniffer = new Sniffer(ext_iface, SnifferType::ST_EXT, bridge_id,
pkt_counter);
    int_sniffer = new Sniffer(int_iface, SnifferType::ST_INT, bridge_id,
pkt_counter);

    ext_sniffer->set_dst(int_sniffer);
    int_sniffer->set_dst(ext_sniffer);
}

InlineSet::~InlineSet()
{
    delete ext_sniffer;
    delete int_sniffer;
}

bool InlineSet::open()
{
    if ( !ext_sniffer->open_live() or !int_sniffer->open_live() )
        return false;

    return true;
}

class ExtToInt : public InlineSet
{
public:
    ExtToInt(const char* ext_iface, const char* int_iface, size_t bridge_id,

```

```

        std::atomic_ullong& pkt_counter)
        : InlineSet(ext_iface, int_iface, bridge_id, pkt_counter)
    { }
    ~ExtToInt() { }

    void live() override;
};

void ExtToInt::live()
{
    const PktDirection dir = ( ext_sniffer->get_type() == ST_EXT ) ? EXT_TO_INT :
INT_TO_EXT;
    DetectorManager::init_pipeline(id, dir);
    Logger::msg("Detectors initialized: bridge #zu(ext->int): count %zu:%s", id,
        DetectorManager::get_pipeline_len(),
    DetectorManager::get_pipeline_names().c_str());

    ext_sniffer->sniff();
    DetectorManager::cleanup_pipeline();
}

class IntToExt : public InlineSet
{
public:
    IntToExt(const char* ext_iface, const char* int_iface, size_t bridge_id,
        std::atomic_ullong& pkt_counter)
        : InlineSet(ext_iface, int_iface, bridge_id, pkt_counter)
    { }
    ~IntToExt() { }

    void live() override;
};

void IntToExt::live()
{
    int_sniffer->sniff();
}

// Live bridge

LiveBridge::LiveBridge(const char* ext_iface, const char* int_iface, int
bridge_id)
    : id(bridge_id),
      pkt_counter(0)
{
    ext_to_int = new ExtToInt(ext_iface, int_iface, bridge_id, pkt_counter);
    int_to_ext = new IntToExt(ext_iface, int_iface, bridge_id, pkt_counter);
}

LiveBridge::~~LiveBridge()
{
    delete ext_to_int;
    delete int_to_ext;

    delete etoi;
    delete itoe;
}

bool LiveBridge::open()
{
    Logger::log("openning bridge: #zu", id);

```

```

    if ( !ext_to_int->open() or !int_to_ext->open() )
    {
        Logger::error("couldn't open bridge: #zu", id);
        return false;
    }

    Logger::log("bridge opened: #zu", id);
    return true;
}

void LiveBridge::live()
{
    ThreadContext::set_state(id, TS_RUNNING);

    etoi = new std::thread(&InlineSet::live, ext_to_int);
    itoe = new std::thread(&InlineSet::live, int_to_ext);

    etoi->detach();
    itoe->detach();
}

```

A.1.7 Файл logger.h

```

#ifndef LOGGER_H
#define LOGGER_H

class Logger
{
public:
    static void msg(const char* fmt, ...) __attribute__((format (printf, 1, 2)));
    static void log(const char* fmt, ...) __attribute__((format (printf, 1, 2)));
    static void error(const char* fmt, ...) __attribute__((format (printf, 1,
2)));
};

#endif // LOGGER_H

```

A.1.8 Файл logger.cc

```

#include "logger.h"

#include <cstdarg>
#include <cstdio>
#include <string>

#define STD_BUF_SIZE 1024

static void printout(FILE* out, const char* fmt, va_list ap)
{
    char buf[STD_BUF_SIZE];
    vsnprintf(buf, sizeof(buf), fmt, ap);
    fputs(buf, out);
}

void Logger::msg(const char* fmt, ...)

```

```

{
    va_list ap;
    va_start(ap, fmt);

    std::string l_fmt = fmt;
    l_fmt += "\n";

    printout(stdout, l_fmt.c_str(), ap);

    va_end(ap);
}

void Logger::log(const char* fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);

    std::string l_fmt = "LOG: ";
    l_fmt += fmt;
    l_fmt += "\n";

    printout(stdout, l_fmt.c_str(), ap);

    va_end(ap);
}

void Logger::error(const char* fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);

    std::string l_fmt = "ERROR: ";
    l_fmt += fmt;
    l_fmt += "\n";

    printout(stderr, l_fmt.c_str(), ap);

    va_end(ap);
}

```

A.1.9 Файл net_defines.h

```

#ifndef NET_DEFINES_H
#define NET_DEFINES_H

// 1 octet == 4 bytes
#define BYTES_IN_OCTET 4

// 8 bytes == 64 bits
#define UDP_HDRLEN 8

enum PktDirection
{
    EXT_TO_INT = 0,
    INT_TO_EXT
};

#endif // NET_DEFINES_H

```

A.1.10 Файл sniffer.h

```

#ifndef SNIFFER_H
#define SNIFFER_H

#include <atomic>
#include <pcap/pcap.h>

enum SnifferType
{
    ST_EXT = 0,
    ST_INT
};

class Sniffer
{
public:
    Sniffer(const char* iface_name, SnifferType type, size_t bridge_id,
            std::atomic_ullong& pkt_counter_);

    ~Sniffer();

    void set_dst(Sniffer* s)
    { dst = s; }

    SnifferType get_type() const
    { return tp; }

    bool open_live();
    void sniff();

private:
    void dispatch(const struct pcap_pkthdr*, const u_char*);
    void accept_pkt(const u_char*, unsigned);

private:
    const char* iface;
    SnifferType tp;
    size_t id;
    std::atomic_ullong& pkt_counter;

    pcap_t* handle = nullptr;
    Sniffer* dst = nullptr;
};

#endif // SNIFFER_H

```

A.1.11 Файл sniffer.cc

```

#include "sniffer.h"

#include "detector_manager.h"
#include "logger.h"
#include "net_defines.h"
#include "thread_context.h"

```

```

Sniffer::Sniffer(const char* iface_name, SnifferType type, size_t bridge_id,
    std::atomic_ullong& pkt_counter_)
    : iface(iface_name),
      tp(type),
      id(bridge_id),
      pkt_counter(pkt_counter_)
{
}

Sniffer::~Sniffer()
{
    if ( handle )
        pcap_close(handle);
}

bool Sniffer::open_live()
{
    char err_buf[PCAP_ERRBUF_SIZE];
    const char* dev = iface;

    // FIXIT: maybe we don't need promiscuous mode is set
    handle = pcap_open_live(dev, BUFSIZ, 1, 1, err_buf);
    if ( !handle )
    {
        Logger::error("couldn't open device: %s: %s", dev, err_buf);
        return false;
    }
    if ( pcap_setnonblock(handle, 1, err_buf) )
    {
        Logger::error("couldn't set non-block state for device: %s: %s", dev,
err_buf);
        return false;
    }
    if ( pcap_setdirection(handle, PCAP_D_IN) == PCAP_ERROR )
    {
        Logger::error("couldn't set direction for device: %s: %s", dev,
pcap_geterr(handle));
        return false;
    }

    Logger::log("device opened: %s", dev);

    return true;
}

void Sniffer::sniff()
{
    while ( ThreadContext::get_state(id) == TS_RUNNING )
    {
        struct pcap_pkthdr frame_hdr;
        if ( const u_char* frame = pcap_next(handle, &frame_hdr) )
        {
            dispatch(&frame_hdr, frame);
        }
    }
}

void Sniffer::dispatch(const struct pcap_pkthdr* frame_hdr, const u_char* frame)
{
    const unsigned int frame_len = frame_hdr->len;
    const unsigned long long pkt_num = ++pkt_counter;
    if ( DetectorManager::execute(frame, frame_len, pkt_num) )

```

```

        dst->accept_pkt(frame, frame_len);
    }

void Sniffer::accept_pkt(const u_char* frame, unsigned frame_len)
{
    if ( pcap_inject(handle, frame, frame_len) == PCAP_ERROR )
    {
        Logger::error("bridge #%zu: sniffer type %s: iface %s: %s", id,
            ( tp ? "internal" : "external" ), iface, pcap_geterr(handle));

        ThreadContext::set_state(id, TS_TERMINATED);
    }
}

```

A.1.12 Файл thread_context.h

```

#ifndef THREAD_CONTEXT_H
#define THREAD_CONTEXT_H

#include <vector>
#include <memory>
#include <mutex>

enum ThreadState
{
    TS_NOT_STARTED = 0,
    TS_TERMINATED,
    TS_STOPPED,
    TS_RUNNING
};

using SlotPair = std::pair<std::unique_ptr<std::mutex>, ThreadState>;

class ThreadContext
{
public:
    static void init_slots(const size_t num);

    static ThreadState get_state(const size_t index)
    { return state_slots[index].second; }

    static void set_state(const size_t index, const ThreadState state);

private:
    static std::vector<SlotPair> state_slots;
};

#endif // THREAD_CONTEXT_H

```

A.1.13 Файл thread_context.cc

```

#include "thread_context.h"

std::vector<SlotPair> ThreadContext::state_slots;

```

```

void ThreadContext::init_slots(const size_t num)
{
    state_slots.reserve(num);
    for ( size_t count = 0; count < num; ++count )
    {
        SlotPair pair;
        pair.first = std::make_unique<std::mutex>();
        pair.second = TS_NOT_STARTED;
        state_slots.push_back(std::move(pair));
    }
}

void ThreadContext::set_state(const size_t index, const ThreadState state)
{
    SlotPair& slot = state_slots[index];

    slot.first->lock();
    slot.second = state;
    slot.first->unlock();
}

```

A.1.14 Файл thread_control.h

```

#ifndef THREAD_CONTROL_H
#define THREAD_CONTROL_H

#include <vector>

class LiveBridge;

class ThreadControl
{
public:
    ThreadControl(size_t num_of_bridges);
    ~ThreadControl();

    bool open_bridges();
    void start_live();

    bool is_ok() const;
    void stop_all() const;

private:
    std::vector<LiveBridge*> bridges;
    size_t count_bridges = 0;
};

#endif // THREAD_CONTROL_H

```

A.1.15 Файл thread_control.cc

```

#include "thread_control.h"

#include "config.h"
#include "inline_set.h"

```

```

#include "thread_context.h"

ThreadControl::ThreadControl(size_t num_of_bridges)
{
    bridges.reserve(num_of_bridges);

    Config* conf = Config::get_instance();
    while ( count_bridges != num_of_bridges )
    {
        bridges.push_back(new LiveBridge(conf->ext_iface.c_str(), conf-
>int_iface.c_str(),
        count_bridges));

        count_bridges++;
    }

    ThreadContext::init_slots(num_of_bridges);
}

ThreadControl::~ThreadControl()
{
    for ( auto& bridge : bridges )
        delete bridge;
}

bool ThreadControl::open_bridges()
{
    for ( auto& bridge : bridges )
    {
        if ( !bridge->open() )
            return false;
    }

    return true;
}

void ThreadControl::start_live()
{
    for ( auto& bridge : bridges )
        bridge->live();
}

bool ThreadControl::is_ok() const
{
    for ( size_t index = 0; index < count_bridges; ++index )
    {
        if ( ThreadContext::get_state(index) == TS_TERMINATED )
            return false;
    }

    return true;
}

void ThreadControl::stop_all() const
{
    for ( size_t index = 0; index < count_bridges; ++index )
        ThreadContext::set_state(index, TS_STOPPED);

    std::chrono::milliseconds ms(500);
    std::this_thread::sleep_for(ms);
}

```

A.1.16 Файл watchdog.h

```
#ifndef WATCHDOG_H
#define WATCHDOG_H

class ThreadControl;

class Watchdog
{
public:
    Watchdog() = default;
    ~Watchdog();

    bool init(int argc, char* argv[]);
    bool init_live_bridges();

    int exec();

private:
    ThreadControl* control = nullptr;
};

#endif // WATCHDOG_H
```

A.1.17 Файл watchdog.cc

```
#include "watchdog.h"

#include <cstring>
#include <thread>

#include "config.h"
#include "logger.h"
#include "thread_control.h"

// FIXIT: make this dynamic
#define NUM_OF_BRIDGES 1

Watchdog::~Watchdog()
{
    delete control;
}

// FIXIT: refactor this into separate help component
#define CLI_HELP "USAGE: watchdog -c <config_file.env> [--help]"

bool Watchdog::init(int argc, char* argv[])
{
    bool parse_err = false;
    bool conf_loaded = false;

    for ( int opt = 1, par_val = 0; opt < argc; ++opt )
    {
        if ( !strcmp(argv[opt], "-c") and ( opt + 1 ) < argc )
        {
            conf_loaded = Config::get_instance()->load_config(argv[opt + 1]);
        }
    }
}
```

```

        ++par_val;

        continue;
    }
    else if ( !strcmp(argv[opt], "--help") )
    {
        Logger::msg(CLI_HELP);
        exit(0);
    }
    else if ( !par_val )
    {
        Logger::error("invalid CLI option provided: %s", argv[opt]);
        parse_err = true;
    }

    --par_val;
}

if ( parse_err )
    Logger::msg("run with '--help' to see available arguments");

if ( !conf_loaded and !parse_err )
    Logger::error("config file not loaded");

return !parse_err and conf_loaded;
}

bool Watchdog::init_live_bridges()
{
    control = new ThreadControl(NUM_OF_BRIDGES);

    if ( !control->open_bridges() )
        return false;

    Logger::log("live bridges initialized");
    return true;
}

int Watchdog::exec()
{
    control->start_live();
    Logger::msg("Watchdog is online\nYour network is under defense now");
    while ( true )
    {
        if ( !control->is_ok() )
        {
            control->stop_all();
            return 5;
        }

        // Iteration timeout 1 sec
        std::chrono::seconds sec(1);
        std::this_thread::sleep_for(sec);
    }
    return 0;
}

```

A.2 Текст програми модулю detectors

A.2.1 Файл detector.h

```

#ifndef DETECTOR_H
#define DETECTOR_H

#include <sys/types.h>

#include "core/net_defines.h"

class Detector
{
public:
    Detector(const char* name, const size_t bridge_id, const PktDirection dir)
        : name(name),
          bridge_id(bridge_id),
          dir(dir)
    { }
    virtual ~Detector() = default;

    virtual bool analyze(const u_char* pkt, const unsigned int pkt_len,
        const unsigned long long pkt_num) = 0;

    const char* get_name() const
    { return name; }

protected:
    const char* name;
    const size_t bridge_id;
    const PktDirection dir;
};

#endif // DETECTOR_H

```

A.2.2 Файл ip_flood_analyzer.h

```

#ifndef IP_FLOOD_ANALYZER_H
#define IP_FLOOD_ANALYZER_H

#include "detector.h"

#include <mutex>
#include <thread>
#include <unordered_map>
#include <vector>

#define ip_flood_analyzer_name "ip_flood_analyzer"

class IpFloodAnalyzer : public Detector
{
public:
    IpFloodAnalyzer(const size_t bridge_id, const PktDirection dir,
        int time_window, short vector_size, float threshold);
    ~IpFloodAnalyzer() = default;

```

```

    bool analyze(const u_char* pkt, const unsigned int pkt_len,
                 const unsigned long long pkt_num) override;

private:
    void insert_ip(uint32_t ip);
    void clear_dict();

    void evaluate_entropy();

    void evaluate();
    void detect_anomaly();

private:
    // <IP adress, emergence count>
    using IpDict = std::unordered_map<uint32_t, long long>;

    IpDict ip_dict;
    std::vector<short> threshold_vec;
    const int analyze_time_window;    // in seconds
    const short threshold_vec_size;

    short vector_pos;
    long long pkt_count;
    bool is_traffic;

    const float entropy_threshold;

    std::mutex mtx;
    std::thread evaluation_thrd;
};

#endif // IP_FLOOD_ANALYZER_H

```

A.2.3 Файл ip_flood_analyzer.cc

```

#include "ip_flood_analyzer.h"

#include <cmath>

#include <arpa/inet.h>
#include <netinet/ether.h>
#include <netinet/ip.h>

#include "core/logger.h"

#define DEFAULT_TIME_WINDOW 2    // in seconds
#define DEFAULT_THRESHOLD_VECTOR_SIZE 15
#define DEFAULT_ENTROPY_THRESHOLD 2.0

#define START_VECTOR_POS 0
#define THRESHOLD_LOW_BORDER -1
#define THRESHOLD_TOP_BORDER 1

IpFloodAnalyzer::IpFloodAnalyzer(const size_t bridge_id, const PktDirection dir,
    int time_window, short vector_size, float threshold)
    : Detector(ip_flood_analyzer_name, bridge_id, dir),
      analyze_time_window(( time_window > 0 ) ? time_window :
    DEFAULT_TIME_WINDOW),

```

```

        threshold_vec_size(( vector_size > 0 ) ? vector_size :
DEFAULT_THRESHOLD_VECTOR_SIZE),
        vector_pos(START_VECTOR_POS),
        pkt_count(0),
        is_traffic(false),
        entropy_threshold(( threshold > 0.0 ) ? threshold :
DEFAULT_ENTROPY_THRESHOLD),
        evaluation_thrd(&IpFloodAnalyzer::evaluate, this)
{ evaluation_thrd.detach(); }

bool IpFloodAnalyzer::analyze(const u_char* pkt, const unsigned int, const
unsigned long long)
{
    const ether_header* ethr_frame = (const ether_header*)pkt;
    uint16_t eth_type = ntohs(ethr_frame->ether_type);

    if ( eth_type == ETHERTYPE_IP )
    {
        const ip* ip_pkt = (const ip*)(pkt + ETHER_HDR_LEN);
        insert_ip(ip_pkt->ip_src.s_addr);
    }

    return true;
}

void IpFloodAnalyzer::insert_ip(uint32_t ip)
{
    if ( !mtx.try_lock() )
        return;

    pkt_count++;
    if ( ip_dict.count(ip) != 0 )
        ip_dict[ip]++;
    else
        ip_dict[ip] = 1;

    mtx.unlock();
}

void IpFloodAnalyzer::evaluate()
{
    threshold_vec.reserve(threshold_vec_size);
    threshold_vec.assign(threshold_vec_size, THRESHOLD_LOW_BORDER);
    while ( true )
    {
        while ( vector_pos < threshold_vec_size )
        {

std::this_thread::sleep_for(std::chrono::seconds(analyze_time_window));

            evaluate_entropy();
            clear_dict();

            vector_pos++;
        }

        detect_anomaly();
        vector_pos = START_VECTOR_POS;
    }
}

void IpFloodAnalyzer::clear_dict()
{

```

```

        std::lock_guard<std::mutex> lock(mtx);
        ip_dict.clear();
        pkt_count = 0;
    }

void IpFloodAnalyzer::evaluate_entropy()
{
    mtx.lock();
    long long total_count = pkt_count;
    IpDict dict = ip_dict;
    mtx.unlock();

    if ( total_count == 0 )
    {
        is_traffic = false;
        return;
    }
    else
        is_traffic = true;

    float entropy = 0.0;
    for ( const auto& ip : dict )
    {
        float ip_prob = (float)ip.second / (float)total_count;
        entropy += -(ip_prob * log2(ip_prob));
    }

    bool thresholded = entropy > entropy_threshold;
    auto& value = threshold_vec[vector_pos];
    if ( thresholded and value != THRESHOLD_TOP_BORDER )
        value++;
    else if ( !thresholded and value != THRESHOLD_LOW_BORDER )
        value--;
}

void IpFloodAnalyzer::detect_anomaly()
{
    if ( !is_traffic )
        return;

    short anomaly_count = 0;
    for ( auto& v : threshold_vec )
    {
        if ( v == THRESHOLD_TOP_BORDER )
            anomaly_count++;
    }
    if ( anomaly_count == threshold_vec_size )
    {
        Logger::msg("bridge #%zu(%s): ip_flood_analyzer: ALERT: DDoS condition
met",
                    bridge_id, ( dir == EXT_TO_INT ? "ext" : "int" ));
    }
}

```

A.2.4 Файл network_analyzer.h

```

#ifndef NETWORK_ANALYZER_H
#define NETWORK_ANALYZER_H

```

```

#include "detector.h"

#include <atomic>

#define network_analyzer_name "network_analyzer"

class NetworkAnalyzer : public Detector
{
public:
    NetworkAnalyzer(const size_t bridge_id, const PktDirection dir,
        const int max_threads);
    ~NetworkAnalyzer() = default;

    bool analyze(const u_char* pkt, const unsigned int pkt_len,
        const unsigned long long pkt_num) override;

private:
    void print(const u_char* pkt, const unsigned long long pkt_num);

private:
    const int max_threads;
    std::atomic_ulong num_threads;

};

#endif // NETWORK_ANALYZER_H

```

A.2.5 Файл network_analyzer.cc

```

#include "network_analyzer.h"

#include <cstring>
#include <iomanip>
#include <sstream>
#include <string>
#include <thread>

#include <arpa/inet.h>
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>

#include "core/logger.h"
#include "core/net_defines.h"

#define DEFAULT_MAX_THREADS 1000

static void print_arp(const ether_arp* pkt, const std::string& prefix_log)
{
    const unsigned short int proto_type = ntohs(pkt->arp_pro);
    const unsigned short int operation_code = ntohs(pkt->arp_op);
    const uint8_t* sha = pkt->arp_sha;
    const uint8_t* spa = pkt->arp_spa;
    const uint8_t* tha = pkt->arp_tha;
    const uint8_t* tpa = pkt->arp_tpa;

    std::stringstream ss;

```

```

std::string pkt_line = prefix_log + ": ";
pkt_line += ether_ntoa((ether_addr*)sha);
pkt_line += " → ";
pkt_line += ether_ntoa((ether_addr*)tha);
pkt_line += " ARP";
ss << "0x" << std::setfill('0') << std::setw(4)
    << std::hex << proto_type;
pkt_line += " PTYPE " + ss.str();

if ( operation_code == ARPOP_REQUEST )
{
    pkt_line += " Who has " + std::to_string(tpa[0]);
    pkt_line += "." + std::to_string(tpa[1]);
    pkt_line += "." + std::to_string(tpa[2]);
    pkt_line += "." + std::to_string(tpa[3]);
    pkt_line += "? ";

    pkt_line += "Tell " + std::to_string(spa[0]);
    pkt_line += "." + std::to_string(spa[1]);
    pkt_line += "." + std::to_string(spa[2]);
    pkt_line += "." + std::to_string(spa[3]);
}
else if ( operation_code == ARPOP_REPLY )
{
    pkt_line += " " + std::to_string(spa[0]);
    pkt_line += "." + std::to_string(spa[1]);
    pkt_line += "." + std::to_string(spa[2]);
    pkt_line += "." + std::to_string(spa[3]);
    pkt_line += " is at ";
    pkt_line += ether_ntoa((ether_addr*)sha);
}
else
    pkt_line += " OPER " + std::to_string(operation_code);

Logger::msg("%s", pkt_line.c_str());
}

static void print_icmp(const icmphdr* hdr, const uint8_t ttl, const std::string&
prefix_log)
{
    std::string pkt_line = prefix_log + " ICMP Echo (ping)";
    uint8_t type = hdr->type;
    uint8_t code = hdr->code;
    uint16_t id = ntohs(hdr->un.echo.id);
    uint16_t seq = ntohs(hdr->un.echo.sequence);

    std::stringstream ss;
    ss << "0x" << std::setfill('0') << std::setw(4) << std::hex;
    if ( type == ICMP_ECHO and code == 0 )
    {
        ss << id;
        pkt_line += " request id=" + ss.str();
        pkt_line += ", seq=" + std::to_string(seq);
        pkt_line += ", ttl=" + std::to_string(ttl);
    }
    else if ( type == ICMP_ECHOREPLY and code == 0 )
    {
        ss << id;
        pkt_line += " reply id=" + ss.str();
        pkt_line += ", seq=" + std::to_string(seq);
        pkt_line += ", ttl=" + std::to_string(ttl);
    }
}

```

```

    Logger::msg("%s", pkt_line.c_str());
}

static void print_tcp(const tcphdr* hdr, const uint16_t pkt_len, const
std::string& prefix_log)
{
    std::string pkt_line = prefix_log + " TCP";

    const uint16_t src_port = ntohs(hdr->source);
    const uint16_t dst_port = ntohs(hdr->dest);
    const uint16_t syn = ntohs(hdr->syn);
    const uint16_t ack = ntohs(hdr->ack);
    const uint16_t fin = ntohs(hdr->fin);
    const uint32_t seq = ntohl(hdr->seq);
    const uint32_t ack_seq = ntohl(hdr->ack_seq);
    const uint16_t win_size = ntohs(hdr->window);
    const uint16_t len = pkt_len - (hdr->doff * BYTES_IN_OCTET);

    pkt_line += " " + std::to_string(src_port);
    pkt_line += " →";
    pkt_line += " " + std::to_string(dst_port);

    pkt_line += " [";
    if ( syn and ack )
        pkt_line += "SYN, ACK";
    else if ( fin and ack )
        pkt_line += "FIN, ACK";
    else if ( ack )
        pkt_line += "ACK";
    else if ( syn )
        pkt_line += "SYN";
    pkt_line += " ]";

    pkt_line += " Seq=" + std::to_string(seq);
    pkt_line += " Ack=" + std::to_string(ack_seq);
    pkt_line += " Win=" + std::to_string(win_size);
    pkt_line += " Len=" + std::to_string(len);

    Logger::msg("%s", pkt_line.c_str());
}

static void print_udp(const udphdr* hdr, const std::string& prefix_log)
{
    std::string pkt_line = prefix_log + " UDP";

    const uint16_t src_port = ntohs(hdr->source);
    const uint16_t dst_port = ntohs(hdr->dest);
    const uint16_t len = ntohs(hdr->len) - UDP_HDRLEN;

    pkt_line += " " + std::to_string(src_port);
    pkt_line += " →";
    pkt_line += " " + std::to_string(dst_port);
    pkt_line += " Len=" + std::to_string(len);

    Logger::msg("%s", pkt_line.c_str());
}

static inline bool is_first_ip_fragment(const uint16_t frag_off)
{ return ( frag_off & IP_DF ) or !( frag_off & IP_MF ); }

static void print_ip(const ip* pkt, const std::string& prefix_log)
{
    std::string ip_pkt_line = prefix_log + ": ";

```

```

ip_pkt_line += inet_ntoa(pkt->ip_src);
ip_pkt_line += " → ";
ip_pkt_line += inet_ntoa(pkt->ip_dst);

const uint16_t id = ntohs(pkt->ip_id);
const uint16_t frag_off = ntohs(pkt->ip_off);
const uint8_t ttl = pkt->ip_ttl;
const uint16_t next_pkt_len = ntohs(pkt->ip_len) - (uint16_t)pkt->ip_hl *
BYTES_IN_OCTET;

const u_char* raw_ip_pkt = (const u_char*)pkt;
ip_pkt_line += " ID=" + std::to_string(id);
switch ( pkt->ip_p )
{
case IPPROTO_ICMP:
{
const icmphdr* icmp_pkt_hdr =
(const icmphdr*)(raw_ip_pkt + (pkt->ip_hl * BYTES_IN_OCTET));

print_icmp(icmp_pkt_hdr, ttl, ip_pkt_line);
break;
}
case IPPROTO_TCP:
{
if ( is_first_ip_fragment(frag_off) )
{
const tcphdr* tcp_pkt_hdr = (const tcphdr*)(raw_ip_pkt + (pkt->ip_hl *
BYTES_IN_OCTET));
print_tcp(tcp_pkt_hdr, next_pkt_len, ip_pkt_line);
}
else
{
const uint16_t fragmented_bits = frag_off & IP_OFFMASK;
ip_pkt_line += " fragmented TCP, Offset=" +
std::to_string(fragmented_bits);
ip_pkt_line += ", Len=" + std::to_string(next_pkt_len);
Logger::msg("%s", ip_pkt_line.c_str());
}
break;
}
case IPPROTO_UDP:
{
if ( is_first_ip_fragment(frag_off) )
{
const udphdr* udp_pkt_hdr = (const udphdr*)(raw_ip_pkt + (pkt->ip_hl *
BYTES_IN_OCTET));
print_udp(udp_pkt_hdr, ip_pkt_line);
}
else
{
const uint16_t fragmented_bits = frag_off & IP_OFFMASK;
ip_pkt_line += " fragmented UDP, Offset=" +
std::to_string(fragmented_bits);
ip_pkt_line += ", Len=" + std::to_string(next_pkt_len);
Logger::msg("%s", ip_pkt_line.c_str());
}
break;
}
}
}

NetworkAnalyzer::NetworkAnalyzer(const size_t bridge_id, const PktDirection dir,

```

```

    const int max_threads)
    : Detector(network_analyzer_name, bridge_id, dir),
      max_threads((max_threads > 0) ? max_threads : DEFAULT_MAX_THREADS),
      num_threads(0)
{ }

bool NetworkAnalyzer::analyze(const u_char* pkt, const unsigned int pkt_len,
                             const unsigned long long pkt_num)
{
    // Print in a separate thread, num of active threads is limited
    if ( num_threads < max_threads )
    {
        num_threads++;
        u_char* buf_pkt = new u_char[pkt_len];
        memcpy((u_char*)buf_pkt, (const u_char*)pkt, pkt_len);
        std::thread print_thrd(&NetworkAnalyzer::print, this, buf_pkt, pkt_num);
        print_thrd.detach();
    }

    return true;
}

void NetworkAnalyzer::print(const u_char* pkt, const unsigned long long pkt_num)
{
    std::string prefix_log = "bridge #" + std::to_string(bridge_id);
    prefix_log += ( dir == EXT_TO_INT ) ? "(ext)" : "(int)";
    prefix_log += ": pkt " + std::to_string(pkt_num);

    const ether_header* ethr_frame = (const ether_header*)pkt;
    uint16_t eth_type = ntohs(ethr_frame->ether_type);

    switch ( eth_type )
    {
    {
    case ETHERTYPE_ARP:
    {
        const ether_arp* arp_pkt = (const ether_arp*)(pkt + ETHER_HDR_LEN);
        print_arp(arp_pkt, prefix_log);
        break;
    }
    case ETHERTYPE_IP:
    {
        const ip* ip_pkt = (const ip*)(pkt + ETHER_HDR_LEN);
        print_ip(ip_pkt, prefix_log);
        break;
    }
    }

    delete[] pkt;
    num_threads--;
}

```

A.2.6 Файл `udp_flood_analyzer.h`

```

#ifndef UDP_FLOOD_ANALYZER_H
#define UDP_FLOOD_ANALYZER_H

#include <mutex>
#include <thread>
#include <unordered_map>

```

```

#include "detector.h"

#define udp_flood_analyzer_name "udp_flood_analyzer"

class UdpFloodAnalyzer : public Detector
{
public:
    enum ActionType
    {
        ALERT = 0,
        BLOCK
    };

public:
    UdpFloodAnalyzer(const size_t bridge_id, const PktDirection dir,
        const int time_window, const long threshold, const std::string&
        action_type);
    ~UdpFloodAnalyzer() = default;

    bool analyze(const u_char* pkt, const unsigned int pkt_len,
        const unsigned long long pkt_num) override;

private:
    bool is_blocked(const uint32_t ip, const uint16_t port);
    void insert_udp(const uint32_t ip, const uint16_t port, const uint16_t
        checksum);
    void clear_dict();

    void evaluate();
    void detect_anomaly();

    void action(const uint32_t ip, const uint16_t port);
    void alert(const uint32_t ip, const uint16_t port);
    void block(const uint32_t ip, const uint16_t port);

private:
    // <IP-address, <PORT, emergence count>>
    using UdpFlowDict = std::unordered_map<uint32_t, std::unordered_map<uint16_t,
        unsigned long>>;
    // <IP-address, <PORT, previous checksum hash>>
    using UdpChecksumDict = std::unordered_map<uint32_t,
        std::unordered_map<uint16_t, uint16_t>>;
    // <IP-address, PORT>
    using UdpBlockedMap = std::unordered_multimap<uint32_t, uint16_t>;

    UdpFlowDict udp_flow_dict;
    UdpChecksumDict checksum_dict;
    UdpBlockedMap blocked_udp;
    const unsigned long threshold_pkt_num;
    const int analyze_time_window;    // in seconds
    ActionType action_type;

    std::mutex mtx_flow, mtx_blocked_list;
    std::thread evaluation_thrd;
};

#endif // UDP_FLOOD_ANALYZER_H

```

A.2.7 Файл `udp_flood_analyzer.cc`

```
#include "udp_flood_analyzer.h"

#include <arpa/inet.h>
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <netinet/udp.h>

#include "core/logger.h"

#define DEFAULT_TIME_WINDOW 2    // in seconds
#define DEFAULT_THRESHOLD_PKT_NUM 30
#define DEFAULT_ACTION_TYPE UdpFloodAnalyzer::ActionType::ALERT

UdpFloodAnalyzer::UdpFloodAnalyzer(const size_t bridge_id, const PktDirection dir,
    const int time_window, const long threshold, const std::string& _action_type)
    : Detector(udp_flood_analyzer_name, bridge_id, dir),
      threshold_pkt_num(( threshold > 0 ) ? threshold :
    DEFAULT_THRESHOLD_PKT_NUM),
      analyze_time_window(( time_window > 0 ) ? time_window :
    DEFAULT_TIME_WINDOW),
      evaluation_thrd(&UdpFloodAnalyzer::evaluate, this)
{
    if ( _action_type == "alert" )
        action_type = ALERT;
    else if ( _action_type == "block" )
        action_type = BLOCK;
    else
        action_type = DEFAULT_ACTION_TYPE;

    evaluation_thrd.detach();
}

bool UdpFloodAnalyzer::analyze(const u_char* pkt, const unsigned int, const
unsigned long long)
{
    const ether_header* ethr_frame = (const ether_header*)pkt;
    uint16_t eth_type = ntohs(ethr_frame->ether_type);

    if ( eth_type == ETHERTYPE_IP )
    {
        const ip* ip_hdr = (const ip*)(pkt + ETHER_HDR_LEN);
        if ( ip_hdr->ip_p == IPPROTO_UDP )
        {
            const udphdr* udp = (const udphdr*)(pkt + ETHER_HDR_LEN
                + (ip_hdr->ip_hl * BYTES_IN_OCTET));

            const uint32_t ip = ip_hdr->ip_src.s_addr;
            const uint16_t port = udp->source;

            if ( is_blocked(ip, port) )
                return false;

            insert_udp(ip, port, udp->check);
        }
    }

    return true;
}
```

```

bool UdpFloodAnalyzer::is_blocked(const uint32_t ip, const uint16_t port)
{
    if ( !mtx_blocked_list.try_lock() )
        return false;

    const auto& it_pair = blocked_udp.equal_range(ip);
    for ( auto it = it_pair.first; it != it_pair.second; it++ )
    {
        if ( it->second == port )
        {
            mtx_blocked_list.unlock();
            return true;
        }
    }

    mtx_blocked_list.unlock();
    return false;
}

void UdpFloodAnalyzer::insert_udp(const uint32_t ip, const uint16_t port, const
uint16_t checksum)
{
    bool is_same_checksum = false;

    if ( !mtx_flow.try_lock() )
        return;

    // Insert and analyze checksum
    if ( checksum_dict.count(ip) != 0 )
    {
        auto& flow = checksum_dict[ip];
        if ( flow.count(port) != 0 )
            is_same_checksum = ( flow[port] == checksum );

        flow[port] = checksum;
    }
    else
        checksum_dict[ip][port] = checksum;

    // Insert UDP flow
    if ( is_same_checksum )
    {
        if ( udp_flow_dict.count(ip) != 0 )
        {
            auto& flow = udp_flow_dict[ip];
            if ( flow.count(port) != 0 )
                flow[port]++;
            else
                flow[port] = 1;
        }
        else
            udp_flow_dict[ip][port] = 1;
    }

    mtx_flow.unlock();
}

void UdpFloodAnalyzer::evaluate()
{
    while ( true )
    {
        std::this_thread::sleep_for(std::chrono::seconds(analyze_time_window));
        detect_anomaly();
    }
}

```

```

        clear_dict();
    }
}

void UdpFloodAnalyzer::clear_dict()
{
    std::lock_guard<std::mutex> lock(mtx_flow);
    udp_flow_dict.clear();
    checksum_dict.clear();
}

void UdpFloodAnalyzer::detect_anomaly()
{
    mtx_flow.lock();
    UdpFlowDict udp_dict = udp_flow_dict;
    mtx_flow.unlock();

    for ( const auto& flow : udp_dict )
    {
        for ( const auto& port : flow.second )
        {
            if ( port.second >= threshold_pkt_num )
                action(flow.first, port.first);
        }
    }
}

void UdpFloodAnalyzer::action(const uint32_t ip, const uint16_t port)
{
    switch ( action_type )
    {
        case ALERT:
            alert(ip, port);
            break;
        case BLOCK:
            block(ip, port);
            break;
    }
}

void UdpFloodAnalyzer::alert(uint32_t ip, const uint16_t port)
{
    Logger::msg("bridge #%zu(%s): udp_flood_analyzer: ALERT: anomaly detected: IP
%s, PORT %d",
        bridge_id, ( dir == EXT_TO_INT ? "ext" : "int" ),
        inet_ntoa(in_addr({ip})), ntohs(port));
}

void UdpFloodAnalyzer::block(const uint32_t ip, const uint16_t port)
{
    mtx_blocked_list.lock();
    blocked_udp.insert(std::make_pair<uint32_t, uint16_t>((uint32_t)ip,
(uint16_t)port));
    mtx_blocked_list.unlock();

    Logger::msg("bridge #%zu(%s): udp_flood_analyzer: BLOCK: anomaly blocked: IP
%s, PORT %d",
        bridge_id, ( dir == EXT_TO_INT ? "ext" : "int" ),
        inet_ntoa(in_addr({ip})), ntohs(port));
}

```

A.3 Файл main.cc

```
#include "core/watchdog.h"

int main(int argc, char* argv[])
{
    Watchdog dog;

    if ( !dog.init(argc, argv) )
        return 1;

    if ( !dog.init_live_bridges() )
        return 2;

    return dog.exec();
}
```

ДОДАТОК Б
Слайди презентації

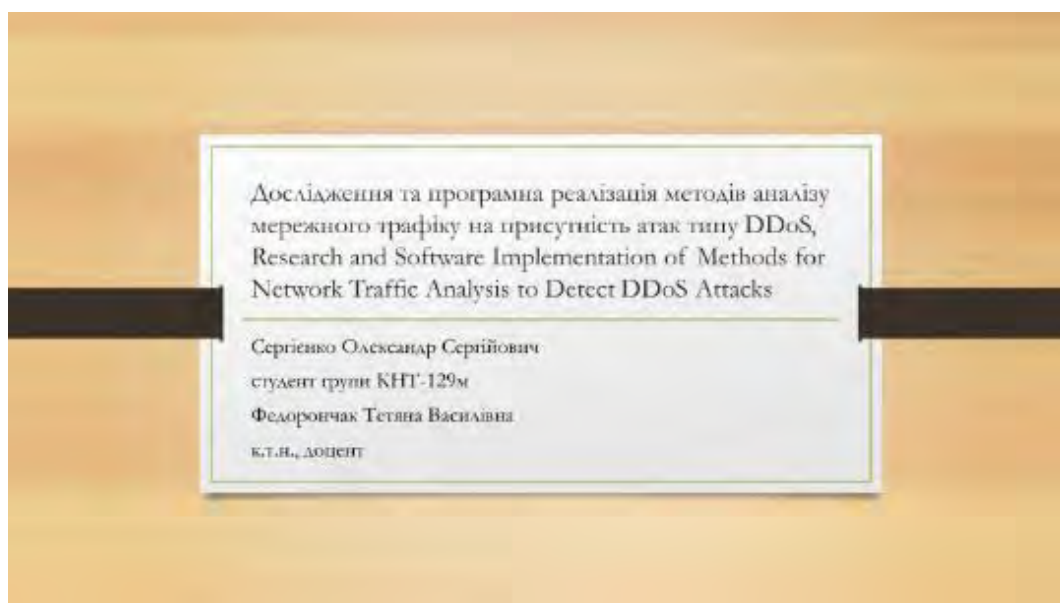


Рисунок Б.1 – Слайд 1

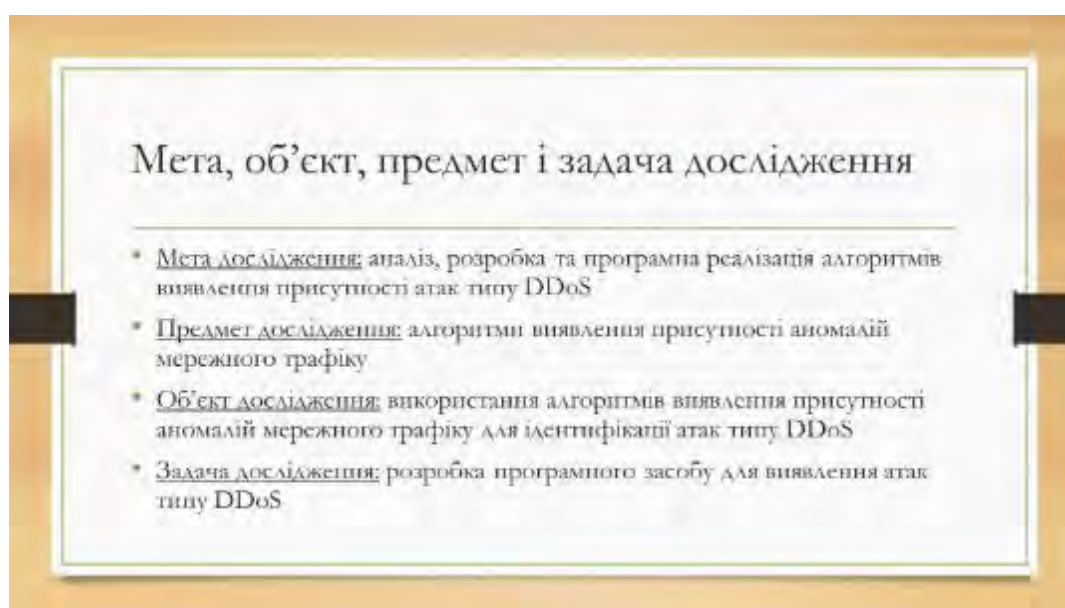


Рисунок Б.2 – Слайд 2



Рисунок Б.3 – Слайд 3



Рисунок Б.4 – Слайд 4

Алгоритм виявлення аномалій мережного трафіку

- Формула для розрахунку ентропії Шеннона:

$$H = - \sum_{i=1}^n p_i \log_2 p_i$$

- Вектор критичних значень:

$$X^T = [-1, -1, \dots, -1, -1]$$

- Оновлення вектору критичних значень на кожній ітерації:

$$X_i^T = \begin{cases} +1, & H_i \geq \omega \\ -1, & H_i < \omega \end{cases}$$

- Умова виявлення присутності аномалій:

$$X^T = [+1, +1, \dots, +1, +1]$$

Рисунок Б.5 – Слайд 5

Зміна значення ентропії Шеннона з плином часу

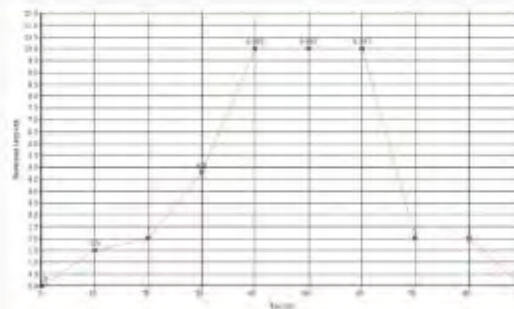


Рисунок Б.6 – Слайд 6

Алгоритм виявлення UDP-флуду

- Заголовок UDP-пакета:

Біти	0-15	16-31
0-31	Порт відправника (src port)	Порт одержувача (dst port)
32-63	Довжина пакету (len)	Контрольна сума (checksum)

- Умова виявлення присутності аномалії:

$$d_N(y_n) = \begin{cases} 0, & (y_n \leq N) \\ 1, & (y_n > N) \end{cases}$$

Рисунок Б.7 – Слайд 7

Схема роботи програми як системного застосунку

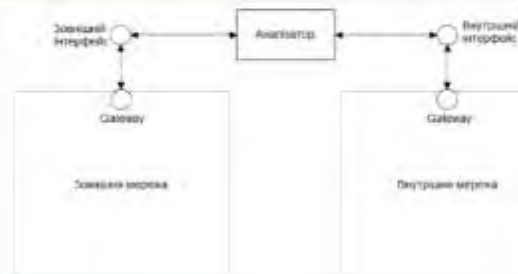


Рисунок Б.8 – Слайд 8

Діаграма послідовностей для опису потокової архітектури програми

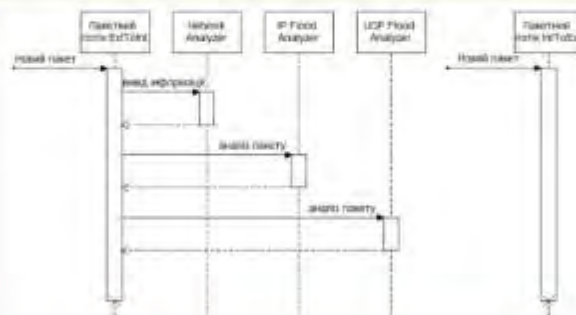
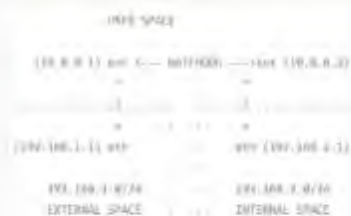


Рисунок Б.11 – Слайд 11

Засоби тестування та схема тестового стенду



- hping3
- LOIC
- Сервер на Python
- curl
- Зразки мережного трафіку (файли .pcap)
- tcpReplay

Рисунок Б.12 – Слайд 12

Вивід пакетної інформації

The screenshot displays a network packet capture analysis tool, likely Wireshark, showing a list of captured packets. The interface is dark-themed. The top section shows the packet list, and the bottom section shows the packet details and packet bytes. The packet list includes columns for No., Time, Source, Destination, Protocol, Length, and Info. The details pane shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, and Hypertext Transfer Protocol. The packet bytes pane shows the raw data of the selected packet in hexadecimal and ASCII.

```

No. 100 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 101 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 102 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 103 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 104 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 105 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 106 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 107 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 108 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 109 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 110 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 111 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 112 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 113 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 114 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 115 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 116 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 117 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 118 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0
No. 119 1.000 192.168.1.101 → 192.168.1.100 [RST] Seq=1000000000 Win=0 Len=0
No. 120 1.000 192.168.1.100 → 192.168.1.101 [RST] Seq=1000000000 Win=0 Len=0

```

Рисунок Б.14 – Слайд 14

Повідомлення про присутність DDoS-атаки

```
alexandr@alexand-laptop: ~/watchdog_install/bin$
File Edit View Search Terminal Help
alexandr@alexand-laptop: ~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
bridge #0(ext): ip_flood_analyzer: ALERT: DDoS condition met
```

Рисунок Б.15 – Слайд 15

Повідомлення про виявлення та блокування UDP-флуду

```
alexandr@alexand-laptop: ~/watchdog_install/bin$ sudo ./watchdog -c conf.env
LOG: config file loaded: conf.env
LOG: opening bridge: #0
LOG: device opened: ext
LOG: device opened: int
LOG: device opened: ext
LOG: device opened: int
LOG: bridge opened: #0
LOG: live bridges initialized
Watchdog is online
Your network is under defense now
Detectors initialized: bridge #0(ext->int): count 2: udp_flood_analyzer ip_flood_analyzer
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 52724
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 53609
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 59810
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 55100
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 56808
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 37372
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 46277
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 43853
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 50750
bridge #0(ext): udp_flood_analyzer: BLOCK: anomaly blocked: IP 192.168.1.1, PORT 48199
```

Рисунок Б.16 – Слайд 16

Результати тестування

- Підтверджено функціональні вимоги
- Підтверджено вимоги до надійності системи
- Робота програми не впливає на асінхронний трафік
- Аналіз зміни ентропії підтверджує правильність та ефективність реалізації алгоритмів
- Вплив роботи програми на мережну затримку є мінімальним (близько 5-6%)

Висновки

- Покращено та успішно реалізовано алгоритм виявлення DDoS-атак на основі розрахунку ентропії Шеннона
- Розроблено та успішно реалізовано алгоритм виявлення UDP-флуду на основі аналізу значень контрольних сум послідовних UDP-пакетів
- Успішно реалізовано програмне забезпечення для виявлення та блокування атак типу DDoS

Рисунок Б.19 – Слайд 19

Наукова новизна та практична цінність

- Оптимізовано та спрощено алгоритм виявлення аномалій мережного трафіку, що базується на основі розрахунку ентропії Шеннона, шляхом використання сталих часових інтервалів аналізу замість фіксованого розміру «пакетного вікна»
- Розроблено алгоритм виявлення UDP-флуду
- Розроблене програмне забезпечення можна використовувати як основу для подальшого дослідження проблеми, оптимізації алгоритмів виявлення аномалій мережного трафіку та для захисту серверів від атак типу DDoS

Рисунок Б.20 – Слайд 20

Кроки для подальшого розвитку

- Використання більш гнучкого формату файлу конфігурації
- Розробка графічного інтерфейсу користувача (Web-додатку віддаленого управління)
- Подальше розширення функціоналу шляхом розробки та реалізації алгоритмів виявлення інших векторів впливу атак типу DDoS

Рисунок Б.21 – Слайд 21